Clustering-based Approximate Answering of Query Result in Large and Distributed Databases

Mounir Bechchi

École Centrale de Nantes **Université de Nantes** **École des Mines de Nantes**

ÉCOLE DOCTORALE STIM

« SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATÉRIAUX »

Année 2009

N° attribué par la bibliothèque

Clustering-based Approximate Answering of Query Result in Large and Distributed Databases

---

THÈSE DE DOCTORAT

Discipline : Informatique

Spécialité : Bases de Données

Présentée
et soutenue publiquement par

Mounir BECHCHI

*Le 15 septembre 2009 à l'*UFR *Sciences & Techniques, Université de Nantes,*
devant le jury ci-dessous

Président	:	Pr. Anne Doucet	LIP6, Université Paris 6
Rapporteurs	:	Mokrane Bouzeghoub, Pr.	PRiSM, Université de Versailles
		Florence Sèdes, Pr.	IRIT, Université Paul Sabatier
Examinateurs	:	Noureddine Mouaddib, Pr.	LINA, Université de Nantes
		Guillaume Raschia, M.C.	LINA, Université de Nantes

Directeur de thèse : Noureddine Mouaddib
Co-encadrant : Guillaume Raschia

Laboratoire: **L**ABORATOIRE D'**I**NFORMATIQUE DE **N**ANTES **A**TLANTIQUE.
UMR CNRS 6241. 2, rue de la Houssinière, BP 92 208 – 44 322 Nantes, CEDEX 3.

N° ED 503-055

CLUSTERING-BASED APPROXIMATE ANSWERING OF QUERY RESULT IN LARGE AND DISTRIBUTED DATABASES

Réponses Approchées de Résultats de Requêtes par Classification dans des Bases de Données Volumineuses et Distribuées

Mounir BECHCHI

⋈

favet neptunus eunti

Université de Nantes

Mounir BECHCHI

Clustering-based Approximate Answering of Query Result in Large and Distributed Databases

IV+XVIII+134 p.

This document was edited with `these-LINA` v. 2.7 LaTeX2e class of the "Association of Young Researchers on Computer Science (I9GIN)" from the University of Nantes (available on : `http://login.irin.sciences.univ-nantes.fr/`). This LaTeX2e class is under the recommendations of the National Education Ministry of Undergraduate and Graduate Studies (circulaire n° 05-094 du 29 March 2005) of the University of Nantes and the Doctoral School of « Technologies de l'Information et des Matériaux(ED-STIM) ».

Print : CorpsTheseMounir.tex – 04/10/2009 – 15:16.

Last class review: these-LINA.cls,v 2.7 2006/09/12 17:18:53 mancheron Exp

Abstract

Database systems are increasingly used for interactive and exploratory data retrieval. In such retrievals, users queries often result in too many answers, so users waste significant time and efforts sifting and sorting through these answers to find the relevant ones. In this thesis, we first propose an efficient and effective algorithm coined Explore-Select-Rearrange Algorithm (*ESRA*), based on the SAINTETIQ model, to quickly provide users with hierarchical clustering schemas of their query results. SAINTETIQ is a domain knowledge-based approach that provides multi-resolution summaries of structured data stored into a database. Each node (or summary) of the hierarchy provided by *ESRA* describes a subset of the result set in a user-friendly form based on domain knowledge. The user then navigates through this hierarchy structure in a top-down fashion, exploring the summaries of interest while ignoring the rest. Experimental results show that the *ESRA* algorithm is efficient and provides well-formed (tight and clearly separated) and well-organized clusters of query results. The *ESRA* algorithm assumes that the summary hierarchy of the queried data is already built using SAINTETIQ and available as input. However, SAINTETIQ requires full access to the data which is going to be summarized. This requirement severely limits the applicability of the *ESRA* algorithm in a distributed environment, where data is distributed across many sites and transmitting the data to a central site is not feasible or even desirable. The second contribution of this thesis is therefore a solution for summarizing distributed data without a prior "unification" of the data sources. We assume that the sources maintain their own summary hierarchies (local models), and we propose new algorithms for merging them into a single final one (global model). An experimental study shows that our merging algorithms result in high quality clustering schemas of the entire distributed data and are very efficient in terms of computational time.

Keywords: Relational databases, Database summaries (the SAINTETIQ model), Clustering of query results, Distributed clustering.

Résumé

Les utilisateurs des bases de données doivent faire face au problème de surcharge d'information lors de l'interrogation de leurs données, qui se traduit par un nombre de réponses trop élevé à des requêtes exploratoires. Pour remédier à ce problème, nous proposons un algorithme efficace et rapide, appelé *ESRA* (Explore-Select-Rearrange Algorithm), qui utilise les résumés SAINTETIQ pré-calculés sur l'ensemble des données pour regrouper les réponses à une requête utilisateur en un ensemble de classes (ou résumés) organisées hiérarchiquement. Chaque classe décrit un sous-ensemble de résultats dont les propriétés sont voisines. L'utilisateur pourra ainsi explorer la hiérarchie pour localiser les données qui l'intéressent et en écarter les autres. Les résultats expérimentaux montrent que l'algorithme *ESRA* est efficace et fournit des classes bien formées (i.e., leur nombre reste faible et elles sont bien séparées). Cependant, le modèle SAINTETIQ, utilisé par l'algorithme *ESRA*, exige que les données soient disponibles sur le serveur des résumés. Cette hypothèse rend inapplicable l'algorithme *ESRA* dans des environnements distribués où il est souvent impossible ou peu souhaitable de rassembler toutes les données sur un même site. Pour remédier à ce problème, nous proposons une collection d'algorithmes qui combinent deux résumés générés localement et de manière autonome sur deux sites distincts pour en produire un seul résumant l'ensemble des données distribuées, sans accéder aux données d'origine. Les résultats expérimentaux montrent que ces algorithmes sont aussi performants que l'approche centralisée (i.e., SAINTETIQ appliqué aux données après regroupement sur un même site) et produisent des hiérarchies très semblables en structure et en qualité à celles produites par l'approche centralisée.

Mots-clés : Base de données relationnelles, Résumés de données (Le modèle SAINTETIQ), Classification des résultats de requêtes, Classification distribuée.

Acknowledgements

I would like to thank, first and foremost, my adviser Prof. Noureddine Mouaddib for his guidance and support throughout my graduate career.

A special thank goes to my co-adviser, Dr. Guillaume Raschia for his patience, understanding, and the stimulating discussions we have had over the years.

I would also like to gratefully acknowledge my colleagues and friends at LINA : Manal El-Dick, Lorraine Goeuriot, Rabab Hayek, Lamiaa Naoum and Amenel Voglozin.

I would like to thank Myriam who shared every nuance of the last five years with me. Her love and support has given me the strength and confidence to complete this endeavor. Finally, I am forever indebted to my family, my mother Zohra, my father Mohammed, my two sisters Fatima and Rajae, and my brother Mohssine, for their understanding, endless love, patience, and encouragement.

Mounir

Table of Contents

List of Tables

— *Body of the Dissertation* —

List of Figures

— *Body of the Dissertation* —

Introduction

With the advent of the World Wide Web, the mass of data daily pouring onto us or at least standing at our free disposal is increasing with an enormous pace. This immense quantity of available data offers us not only an unprecedented wealth of riches and possibilities, it also introduces a new challenge : retrieving relevant data accurately and quickly without being distracted by irrelevant data. In fact, searching the World Wide Web is often like finding a needle in a haystack since the right information users want likely would remain buried under a mass of irrelevancies. This thesis presents new techniques to meet this challenge. In the following, we describe the context, the motivation, the main contributions and the structure of the thesis.

Context

With the rapid development of the World Wide Web, more and more accessible databases are available online. Such databases are usually called "Web databases". From this angle, the World Wide Web can be divided into two parts [Ber01] : Surface Web and Deep Web (or Hidden Web). The Surface Web refers to the static Web pages which can be accessed directly through URL links such as personal homepages, while the Deep Web refers to the Web pages which are hidden behind query interfaces (usually an HTML form) and are dynamically generated by Web databases such as www.amazon.com, www.eBay.com, www.cars.com and www.realestate.com. The Deep Web is becoming a very important information resource. In fact, a recent survey [Ber01] indicates that the Web has about 550 billion Web pages and only about 1% of them are in the Surface Web while the rest is in the Deep Web.

There are mainly two categories of Web databases in the Deep Web. One is the unstructured Web database that returns unstructured data objects, e.g., text or image, in response to user queries. The other is the structured (or relational) Web database that returns data objects as structured records with multiple fields, e.g., a query against the Web database of the book vendor www.amazon.com returns the title, authors, publisher, price, ISBN, etc. of qualifying books. In this thesis we mainly focus on structured Web databases for the following reasons. Firstly, a recent study [CHL+04] indicates that 80% of online databases are structured databases. Secondly, the large amounts of structured data, amassed in structured Web databases, are of greater interest to users who are looking for information in specific domains such as book,

real estate, automobile, or air travel. For simplicity, we use the term "Web database" in the following sections to refer to "structured Web database".

Data residing in Web databases are guarded by query interfaces and, therefore, beyond the reach of traditional search engines. In fact, standard search engines such as Google that gather information by crawling the Web cannot index the content of these Web databases, as the engine would have to fill meaningful inputs in query interfaces for retrieving such dynamically generated data. This means that the main way people access Web databases is through their query interfaces. However, while there are myriad useful Web databases for any domain of interest (e.g., www.amazon.com, www.barnesandnoble.com, www.alapage.com, etc. for books), users often have difficulty in first finding the right Web databases to query in order to retrieve the desired information. Furthermore, these Web databases provide similar or related content but with varied coverage and querying capabilities. Consequently, users often need to interact with multiple Web databases, understand their query syntaxes, formulate separate queries, and compile query results from different Web databases. This can be an extremely time-consuming and labor-intensive process. To enable a "one-stop" access for such a wealthy amount of high quality data, there are two major kinds of design choices among others[1] : "warehouse" and "metasearch" approaches (see Figure 1). Both approaches take a set of Web databases related to a particular domain, and they provide people a uniform access to them.

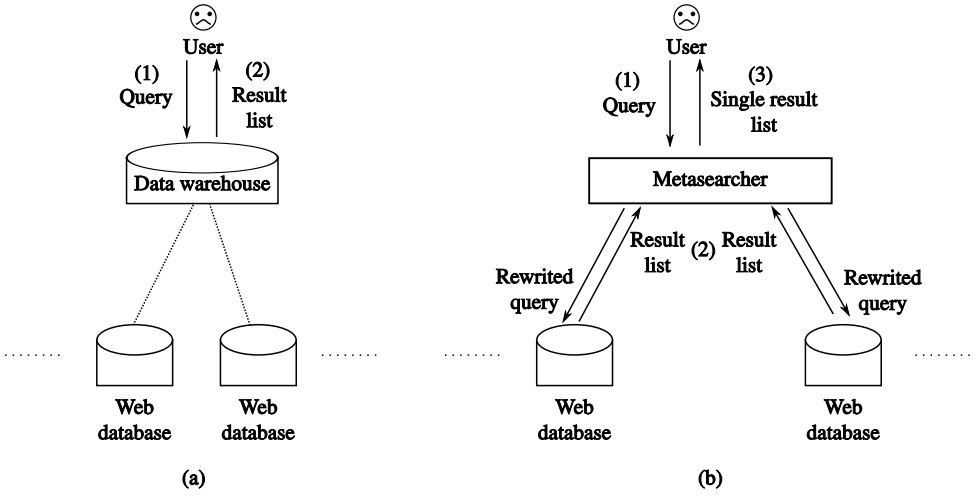


Figure 1 – (a) The warehouse approach (b) The metasearch approach

[1]Some Deep Web portal services provide Deep Web directories which classify Web databases in some taxonomies. For example, CompletePlanet (www.completeplanet.com), the biggest Deep Web directory, has collected more than 7000 Web databases and classified them into 42 topics.

In the warehouse approach, e.g., Google Shopping[2] and MSN Shopping[3], the data is extracted from multiple Web databases in an offline manner. Then, this extracted data is stored in a centralized database system (or repository) that is responsible of query processing, i.e., user queries are answered solely based on the data stored in the centralized warehouse.

On the other hand, the metasearch approach, e.g., Wise-Integrator [HMYW04] and MetaQuerier [CHZ05], provides users with an integrated interface over the query interfaces of multiple Web databases. The metasearcher performs three main tasks to process a user's query filled in the integrated interface : it determines the appropriate Web databases to invoke for a given user query (database selection), it rewrites the query in a suitable form for each selected Web database (query rewriting), and finally it retrieves and merges the results from these Web databases (result merging) before returning them to the user.

Each of the above approaches has its specific advantages and disadvantages. For instance, the warehouse approach has the advantage of efficient query processing, but its main weakness is that the data warehouse needs to be updated frequently to reflect the changes (i.e., updates) in the integrated Web databases. This disadvantage of using a centralized data repository is overcome by the metasearch approach since user queries are executed directly at the integrated Web databases and the results are merged on the fly to answer a query. The main disadvantage of this latter approach, however, is that the user has to wait a long time before the results can be displayed.

Motivation

In recent years, there has been a great deal of interest in developing effective and efficient techniques for exploratory search in Web databases. This interest stems from the fact that the increased visibility of these structured data repositories made them accessible to a large number of lay users, typically lacking a clear view of their content, moreover, not even having a particular item in mind. Rather, they are attempting to discover potentially useful items. In such retrievals, whether the warehouse approach or the metasearch approach is followed, queries often result in too many answers. Not all the retrieved items are relevant to the user. Unfortunately, she/he often needs to examine all or most of them to find the interesting ones. This too-many-answers phenomenon is commonly referred to as "information overload" - a state in which the amount of information that merits attention exceeds an individual's ability to process it [SV98].

[2] www.google.com/advanced_product_search

[3] www.shopping.msn.com

Information overload often happens when the user is not certain of what she/he is looking for, i.e., she/he has a vague and poorly defined information need or *retrieval goal*. In such a situation, she/he generally pose a broad query in the beginning to avoid exclusion of potentially interesting results and next, she/he starts browsing the answer looking for something interesting. Information overload makes it hard for the user to separate the interesting items from the uninteresting ones, thereby leading to potential decision paralysis and a wastage of time and effort. The dangers of information overload are not to be underestimated and are well illustrated by buzzwords such as "Infoglut" [All92], "Information Fatigue Syndrome" [Lew96], "TechnoStress" [WR97], "Data Smog" [She97], "Data Asphyxiation" [Win98] and "Information Pollution" [Nie03]. Whole new fields of psychology have been opened just to deal with the information overload and with our difficulties and perplexity resulting therefrom.

In the context of today's Web databases, automated ranking and clustering of query results are used to reduce information overload. Automated ranking-based techniques first seek to clarify or approximate the user's retrieval goal. Then, they assign a score to each answer, representing the extent to which it is relevant to the approximated retrieval goal. Finally, the user is provided with a ranked list, in descending order of relevance, of either all query results or only a top-k subset. In contrast, clustering-based techniques assist the user to clarify or refine the retrieval goal instead of trying to learn it. They consist in dividing the query result set into dissimilar groups (or clusters) of similar items, allowing users to select and explore groups that are of interest to them while ignoring the rest. However, both of these techniques present two major problems :

1. the first is related to *relevance*. With regard to automated ranking-based techniques, the relevance of the results highly depends on their ability to accurately capture the user's retrieval goal, which is not an obvious task. Furthermore, such techniques also bring the disadvantage of match homogeneity, i.e., the user is often required to go through a large number of similar results before finding the next different result. With regard to clustering-based techniques, there is no guarantee that the resulting clusters will match the meaningful groups that a user may expect. In fact, most clustering techniques seek to only maximize (or minimize) some statistical properties of the clusters (such as the size and compactness of each cluster and the separation of clusters relative to each other) ;
2. the second is related to *scalability*. Both ranking and clustering are performed on query results and consequently occur at query time. Thus, the overhead time cost is an open critical issue for such *a posteriori* tasks. This problem is further exacerbated in the case of the metasearch approach due to the communication delay required to gather the results

from multiple Web databases located at different sites connected by a network.

As a consequence, investigations into new effective and efficient techniques for dealing with the huge amount of data reachable by querying Web databases remain topical.

Contribution

In this thesis we investigate a simple but useful strategy to alleviate the two previously mentioned problems (i.e., relevance and scalability), in both the warehouse and the metasearch approaches. Generally speaking, we propose novel algorithms to quickly provide users with approximate but very convenient for decision support, representations of their query results. These algorithms operate on pre-computed knowledge-based summaries of the data, instead of the data itself. The underlying summarization technique used in this thesis is the SAINTETIQ model [RM02, SPRM05], which is a domain knowledge-based approach that enables summarization and classification of structured data stored into a database. SAINTETIQ first uses prior domain knowledge to transform raw data into high-level representations (summaries) that are more suitable for decision making. Then it applies a hierarchical clustering algorithm on these summaries to provide multi-resolution summaries (i.e., summary hierarchy) that represent the database content at different abstraction levels.

The main contributions of this thesis are the following.

In the warehouse approach, our first contribution [BVRM08] consists in proposing an effective and efficient algorithm coined Explore-Select-Rearrange (*ESRA*) that provides users with hierarchical clustering schemas of their query results. Given a user query, the *ESRA* algorithm (i) explores the summary hierarchy (computed offline using the SAINTETIQ model) of the whole data stored in the warehouse ; (ii) selects the most relevant summaries to that query ; (iii) rearranges them in a hierarchical structure based on the structure of the pre-computed summary hierarchy and (iv) returns the resulting hierarchy to the user. Each node (or summary) of the resulting hierarchy describes a subset of the result set in a user-friendly form based on domain knowledge. The user then navigates through this hierarchy structure in a top-down fashion, exploring the summaries of interest while ignoring the rest. Note that the data warehouse is accessed only when the user requests to download (Upload) the original data that a potentially relevant summary describes. The above process is sketched in Figure 2. Experimental results show that the *ESRA* algorithm is efficient and provides well-formed (tight and clearly separated) and well-organized clusters of query results. Thus, it is very helpful to users who have vague and poorly defined retrieval goals or are interested in browsing through a set of items to explore

what choices are available.

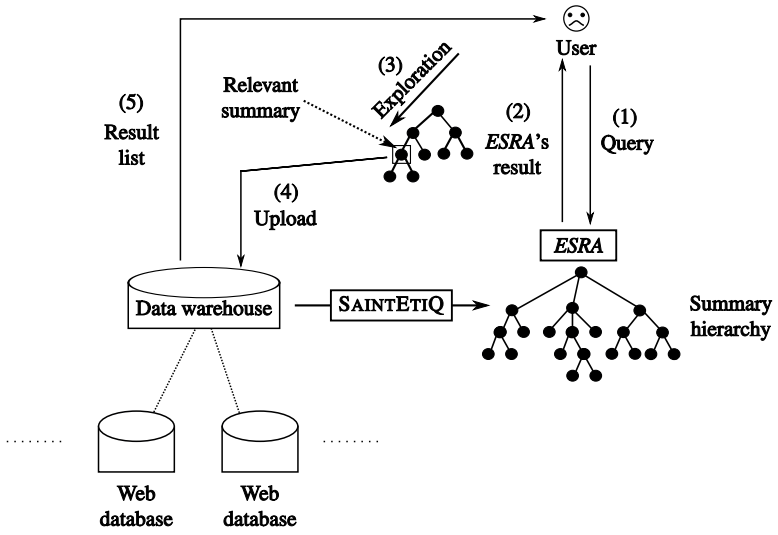


Figure 2 – *ESRA* architecture for the warehouse approach

The *ESRA* algorithm assumes that the summary hierarchy of the queried data is already built using SAINTETIQ and available as input. However, the SAINTETIQ model requires full access to the data which is going to be summarized, i.e., all data has to be located at the site where it is summarized. This requirement severely limits the applicability of the *ESRA* algorithm in the metasearch approach since collecting data at one location is against the main motivation (i.e., decentralized management of distributed data) behind this approach. The second contribution [BRM07, BRM08] of this thesis is a solution for summarizing distributed data without a prior "unification" of the data sources. We assume that the underlying Web databases maintain their own summary hierarchies (local models), and we propose new algorithms for merging them into a single final one (global model). The main idea of these algorithms consists in applying innovative merges on the local models to provide the global one without scanning the raw data. The global summary hierarchy can then be used, as in the warehouse approach, to answer a query submitted to the metasearcher. Note that in the case where Web databases use the *ESRA* algorithm to handle queries, our merging algorithms can be directly applied to local *ESRA*'s results (partial results) to provide the user with a single, global hierarchical clustering schema of the global result set (see Figure 3). This architecture is consistent since *ESRA*'s results are also summary hierarchies. An experimental study shows that our merging algorithms result in high quality clustering schemas of the entire distributed data and are very efficient in terms of computational time.

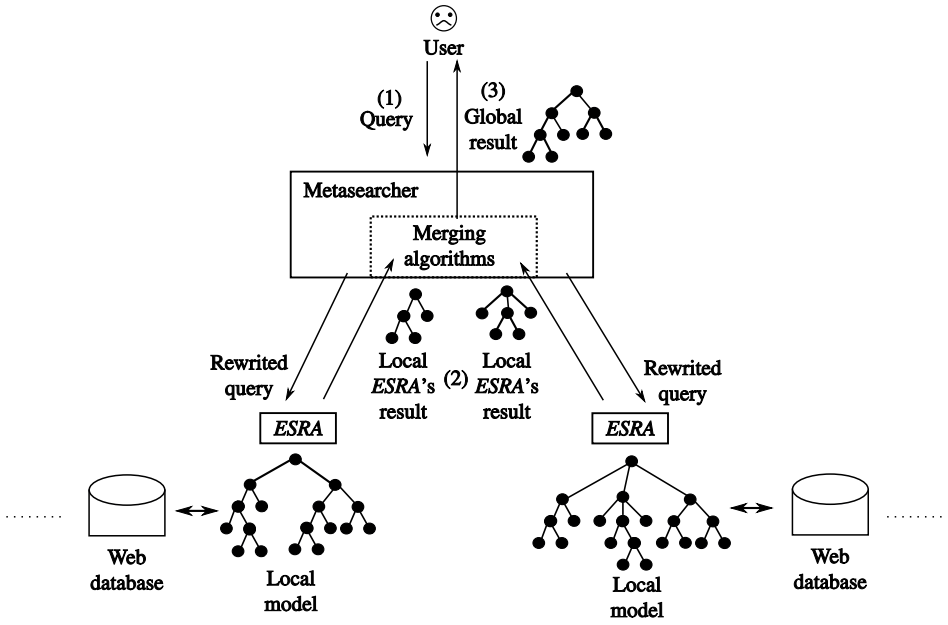


Figure 3 – *ESRA* architecture for the metasearch approach

Roadmap

The rest of this thesis is organized as follows. In Chapter 1, we first survey techniques that have been proposed in the literature to provide users with effective and efficient ways to access Web databases, and then propose a categorization of these techniques based on the problem that they are supposed to address. Then, in Chapter 2 we present the *ESRA* algorithm and the query answering system that supports *ESRA*-based summary hierarchies. Our solutions for summarizing distributed data are presented in Chapter 3. Finally, we conclude and discuss future directions of research.

CHAPTER 1

Advanced Query Processing in Databases

Introduction

As internet becomes ubiquitous, many people are searching their favorite houses, cars, movies, cameras, restaurants, and so on over the Web. Most web sites[4] use databases to store their data and provide SQL-based query interfaces for users to interact with databases. Database systems provide well-maintained and high-quality structured data. However, unlike Web search engines that take a few keywords, look up the index and provide a listing of best-matching web pages, they expect users to know the name of the relation to query, the field to look in, and at times even the field type [Nam05]. Moreover, database query processing models have always assumed that the user knows what she/he wants and is able to formulate a query that accurately expresses her/his needs. Therefore, most database systems have always used a boolean model of query processing where there is a set of answer tuples that exactly satisfy all the constraints of the query and thus are equally relevant to the query.

While extremely useful for the expert user, the above retrieval model is inadequate for lay users who cannot articulate the perfect query for their needs - either their queries are very specific, resulting in no (or too few) answers, or are very broad, resulting in too many answers. Hence, to obtain a satisfactory answer from a database, users must reformulate their queries a number of times before they can obtain a satisfactory answer. However, this process is frustrating, tedious and time-consuming.

In this chapter, we review and discuss several research efforts that have attempted to handle the dual issues of empty and many answers. Although the list of approaches described below is not exhaustive, it provides a representative list of some commonly used approaches. The remaining of this chapter is organized as follows. In Section 1.1, we review a number of ap-

[4] A July 2000 study [Ber01] estimated $96,000$ relational databases were online and the number has increased by 7 times in 2004 [CHL+04].

proaches for handling the *empty-answer* problem, that is, the problem of not being able to provide the user with any data fitting her/his query. Section 1.2 presents some works addressing the *many-answers* problem, i.e., the situation where the user query results in overabundant answers. Then, in Section 1.3 we give an overview of flexible and user-friendly querying techniques, the main objective of which is to provide intelligent interfaces to access databases in a more human-oriented fashion and hence diminish the risk of both empty and many answers. Finally, a discussion is presented in Section 1.4.

1.1 Handling the *Empty-Answer* Problem

Most probably, one has encountered answers like "no houses, hotels, vehicles, flights, etc. could be found that matched your criteria ; please try again with different choices". The case of repeatedly receiving empty query result turns out to be extremely disappointing to the user, and it is even more harmful for the e-merchant.

This problem, which is known as *empty-answer* problem, happens when the user submits a very restrictive query. A simple way to remedy this problem is to retry a particular query repeatedly with alternative values of certain conditions until obtaining satisfactory answers from a database. This solution, however, can be applied only if the user is aware of the close alternatives, otherwise it is infeasible (especially for users who lack knowledge about the contents of the database they wish to access). Many techniques are proposed to overcome this problem, namely query relaxation (Section 1.1.1) and similarity based search (Section 1.1.2).

1.1.1 Query Relaxation

Query relaxation aims to modify the failed query to provide the user with some alternative answers or at least to identify the cause of the failure, rather than just to report the empty result. A database system with such capability is also known as a cooperative information system [GGM94].

Consider a database `EmpDB` with information on employees, including their `Name`, `Age`, `Gender`, `Salary`, `Job` and `Department`, and a query "*get all employees who make less than* $15K€$ *and work in the R&D department*". Note that this query may fail for two different reasons [Mot86b] : either no employee in R&D department makes less than $15K€$ or the company does not have an R&D department. The former is a genuine null answer (i.e., the null answer is appropriate since query fails to match any data), while the latter is a fake null answer (i.e., it is due to the erroneous presupposition that the company has an R&D department).

Unfortunately, current database systems will simply return a null answer. Clearly, a fake null is unsatisfactory, since it may be interpreted by the user as a genuine null and consequently also as an affirmation of the user's presupposition (i.e., the company has an R&D department). Suppose asking this query to a human expert. In the first case, the response would be "there are no employees in the R&D department who earn less than 15K€ but there are some who make less than 20K€". However, the response in the second scenario would be "there is no R&D department".

The first system with such human behavior was developed by Kaplan (CO-OP[5] [Kap83]) and was designed for natural language interaction. The main idea of CO-OP is to follow up a query that failed with several more general queries (i.e., the query with some of its conditions relaxed). If even these general queries fail, then the conclusion is that some of the presuppositions of the user who composed the original query are erroneous. If all these general queries succeed, a query that fails produces a genuine null. Furthermore, assume q' and q'' are both relaxations of the failed query q, but q'' is more general than q'. If both succeed, then the partial answer returned by q' is better (i.e., the best the system could do to satisfy the initial query q). If both fail, then the erroneous presupposition indicated by q'' is stronger. This leads to the conclusion that only Minimal Generalizations that Succeed (MGSs) and maXimal Generalizations that Fail (XGFs) are significant. Indeed, XGFs provide explanation for the failure and some assistance for relaxing the query into a non-failing query, whereas MGSs produce alternative answers to the failing query.

Since then, several systems that adapt CO-OP's techniques to relational databases have been proposed, including SEAVE [Mot86b, Mot86a], CoBase [CCH94, CYC+96] and Godfrey's system [God97]. These systems differ only in the way they perform generalizations. SEAVE [Mot86a, Mot86b] considers all possible generalizations of the query. CoBase [CCH94, CYC+96] uses prior knowledge of the domain to guide the generalization process. Godfrey's system [God97] generalizes the query by only removing some of its conditions.

SEAVE

Given a query that fails, SEAVE[6] [Mot86b, Mot86a] constructs a lattice of its generalized queries and uses it to find all MGSs/XGFs. The query generalizations are obtained by relaxing to a degree some of the conditions in the query.

[5] A COOPerative Query System.

[6] Supposition Extraction And VErification.

As an example (cf. [Mot86b]), consider the following query q over the Employees' relation `EmpDB` :

$$
\begin{array}{rcll}
 & & \text{SELECT} & * \\
 & & \text{FROM} & \texttt{EmpDB} \\
q & \equiv & \text{WHERE} & \texttt{Age} \leq 30 \\
 & & \text{AND} & \texttt{Gender} = \text{F} \\
 & & \text{AND} & \texttt{Salary} \geq 40\text{K}\in
\end{array}
$$

Figure 1.1 shows a portion of the lattice of relaxations of q generated by SEAVE, where nodes indicate generalizations (or presuppositions) and arcs indicate generalization relationships. (x, y, z) denotes a query which returns the employees whose age is under x, whose sex is y, and whose yearly salary is at least z. The symbol * indicates any value ; once it appears in a query, this query cannot be generalized any further. Assume in this example that q fails and all its relaxed queries are successful queries except those that are marked $\surd$ in Figure 1.1 (i.e., q_1, q_3 and q_8). Thus, the failed relaxed queries q_1 and q_3 are XGFs, whereas the successful relaxed queries q_2, q_4, q_6 and q_{15} are MGSs. The queries q_2, q_4, q_6 and q_{15} produce alternative answers to the original query q : (q_2) all employees under 30 who earn at least 40K€ ; (q_4) all female employees under 32 who earn at least 40K€ ; (q_6) all female employees under 31 who earn at least 39K€ ; and (q_{15}) all female employees under 30 who earn at least 37K€ . These answers can be delivered by SEAVE to the user as "the best it could do" to satisfy the query q.

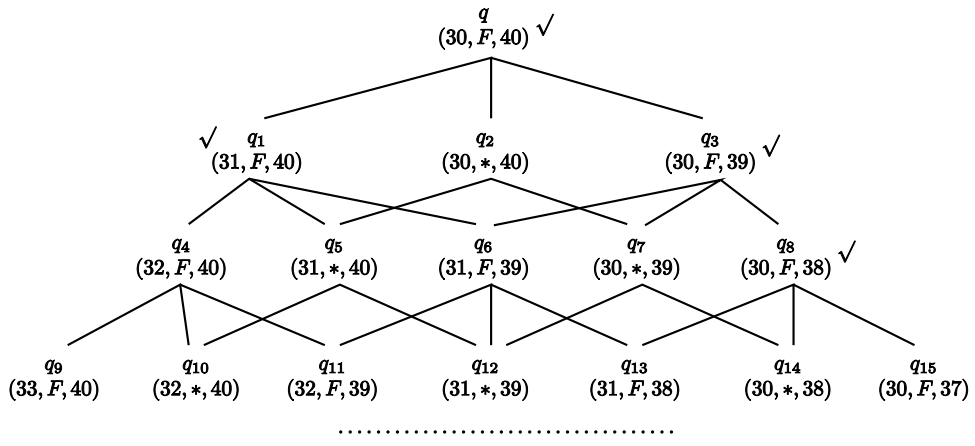


Figure 1.1 – Lattice of generalized queries of q (cf. [Mot86b])

The main drawback of SEAVE is its high computational cost, which comes from computing and testing a large number of generalizations (i.e., various combinations of the values of attributes) to identify MGSs/XGFs.

CoBase

The CoBase[7] [CCH94, CYC+96] system augments the database with Type Abstraction Hierarchies (TAHs) to control the query generalization process. A TAH represents attribute values at different levels of granularity. The higher levels of the hierarchy provide a more abstract data representation than the lower levels (or attribute values). Figure 1.2 shows an example of TAH for attribute `Salary` in which unique salary values are replaced by qualitative ranges of high, medium, or low. CoBase automatically generates the TAHs by clustering all the tuples in the database [CCHY96, MC93]. To relax a failing query, CoBase uses some types of TAH-based operators such as generalization (moving up the TAH) and specialization (moving down the TAH). For example, based on the type abstraction hierarchy given in Figure 1.2, the condition '`Salary` $= 20$K€' could be generalized (i.e., move-up operator) to '`Salary` = medium'.

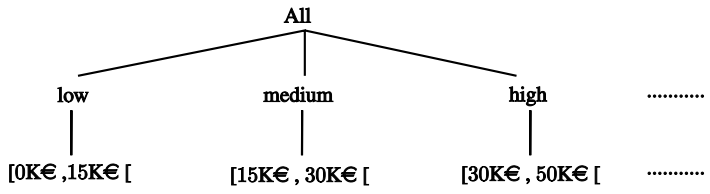


Figure 1.2 – TAH defined on the attribute `Salary`

CoBase considerably reduces the number of generalizations to be tested. Note that how close the results are to the user's initial expectations depends on the TAHs used.

Godfrey's System

In [God97], Godfrey proposed to generalize the user failed query by just removing some of its conditions. Thus, instead of searching all MGSs/XGFs, the proposed system looks for all maXimal Succeeding and Minimal Failing Sub-queries (XSSs and MFSs, respectively) of the failed query. The author also proves that this problem is NP-hard. Indeed, the size of the search space grows exponentially with the number of attributes used in the failed query, i.e., if a query involves m attributes, there exist $(2^m - 2)$ sub-queries that have to be examined, disregarding the query itself and the empty query $\emptyset$. For instance, the lattice of the possible sub-queries of q is illustrated in Figure 1.3, using the same notation as in Figure 1.1.

Hence, recently some heuristics [Mus04, ML05, NK04] have been proposed to prune the search space. In [Mus04] and [ML05], Muslea et al. respectively used decision tree [Qui93]

[7] A Cooperative DataBase System

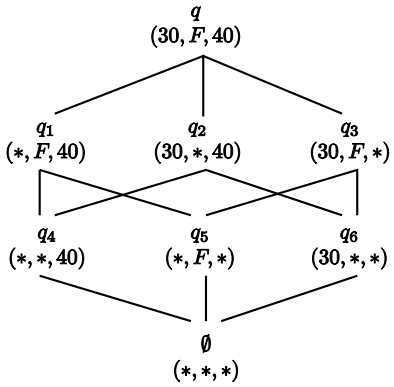


Figure 1.3 – Lattice of possible sub-queries of q

and Bayesian network [FGL98] learning techniques on a randomly-chosen subset of the target database to identify potential relaxations (or sub-queries) of the failing query to be tested. Then, they use nearest-neighbor techniques to find the relaxed query that is the most similar to the failing query. Nambiar et al. [NK04] employed approximate functional dependency [KM92] to get the importance degree of the schema attributes in a database, according to which the order of the relaxed attributes is specified. The data samples used to compute the importance degree are also chosen randomly. Note that both Muslea et al's [Mus04, ML05] and Nambiar et al's [NK04] approaches reduce the search space considerably, but the output relaxed queries, while succeed, are not necessary XSSs.

1.1.2 Similarity Search

A similarity-based search uses a notion of non-exact matching. In other words, when available data stored in the database do not exactly match a user's query, database records which best fit (i.e., which are the most similar to) the query are retrieved and ranked according to their similarity to the query.

Consider a tourist looking for a hotel, with a rent price at $100€$ a day, and located in the city center. She/he will unfortunately fail to find such a hotel by means of the traditional database systems if the city center does not have any hotel rented at that price. However, a similarity-based search system would return the most similar hotels (i.e., hotels having attributes values very close to those specified in the query) instead of the empty result set. In fact, the user might accept a hotel near the city center and the rent price can also be a little lower or higher $100€$ per day.

In the literature, several approaches have been proposed to enhance conventional databases

with similarity search capabilities, such as ARES [IH86], VAGUE [Mot88], IQE [NK03] and Nearest-Neighbors [RKV95]. The former three approaches [IH86, Mot88, NK03] make use of an explicit operator 'similar-to', which extends the usual equality. The latter [RKV95] views the records in the database as points in a multidimensional space and the queries about these records are transformed into the queries over this set of points.

ARES

ARES[8] [IH86] is the first system that has addressed the basic issue of similarity matching. It introduces a new operator named 'similar-to' and denoted $\approx$, meaning "approximately equal to". $\approx$ can be used as a comparison operator inside queries instead of the usual equality operator ($=$) in order to express vague conditions, e.g., $A \approx v$ will select values of an attribute A that are similar to a constant v. The interpretation of $\approx$ is based on dissimilarity relations tied to each domain. A dissimilarity relation, $\mathrm{DR}_A(A_1, A_2, Distance)$, on the domain D_A of attribute A contains triples of the form $(v_1, v_2, dist)$, where $v_1 \in D_A$, $v_2 \in D_A$ and $dist$ represents the distance (dissimilarity) value between v_1 and v_2 (a smaller value means v_1 and v_2 are more similar). Table 1.1 illustrates an example dissimilarity relation for the attribute `Job` of a relation `EmpDB`.

Table 1.1 – Dissimilarity relation defined on the attribute `Job`

Job_1	Job_2	$Distance$
Student	PhD. Student	1
Student	Supervisor	3
Supervisor	PhD. Student	2
...	...	...

In a given query, which contains vague conditions (i.e., conditions involving the similarity operator $\approx$), the following process takes place. First of all, for each vague condition, the user gives a maximum accepted distance value. ARES then accesses dissimilarity relations to produce a boolean query which will be processed by a conventional database system. For example, the vague condition $A \approx v$ is transformed to the boolean one $A \in \{x \in D_A | (v, x, dist) \in \mathrm{DR}_A \wedge dist \leq \tau\}$, where τ is the maximum allowed distance given by the user on D_A. In other words, x and v are considered somewhat close as far as $dist \leq \tau$. The produced query

[8] Associative Information REtrieval System

will then select acceptable tuples for which a global distance is calculated, by summing up the elementary distances tied to each vague condition in the query. Finally, the tuples are sorted in ascending order according to their global distance (dissimilarity) values and the system will output as many tuples as possible within the limit that has been specified by the user.

The main drawback of ARES is its high storage and maintenance costs of dissimilarity relations : each dissimilarity relation needs m^2 entries with respect to m different attribute values in the corresponding conventional relation ; and when a new attribute value is added, $2m + 1$ additional entries are necessary for the corresponding dissimilarity relation [DKW02]. Moreover, ARES does not allow defining dissimilarity between attribute values for infinite domains because the dissimilarities can only be defined by means of tables.

VAGUE

VAGUE[9] [Mot88] is a system that resembles ARES in its overall goals. It is an extension to the relational data model with data metrics and the SQL language with a comparator $\approx$. Indeed, each attribute domain D is endowed with a metric $\mathcal{M}_D$ to define distance (dissimilarity) between its values. $\mathcal{M}_D$ is a mapping from the cartesian product $D \times D$ to the set of non-negative reals which is :

- ***reflexive***, i.e., $\mathcal{M}_D(x, x) = 0$, for every value x in D ;
- ***symmetric***, i.e., $\mathcal{M}_D(x, y) = \mathcal{M}_D(y, x)$, for all values x and y in D ; and
- ***transitive***, i.e., $\mathcal{M}_D(x, y) \leq \mathcal{M}_D(x, z) + \mathcal{M}_D(z, y)$, for all values x, y and z in D.

Furthermore, $\mathcal{M}_D$ is provided with a radius r. This notion is very similar to the maximum dissimilarity allowed in ARES. Thus, two values v_1 and v_2 in D are considered to be similar if $\mathcal{M}_D(v_1, v_2) \leq r$. During query processing, each vague condition expressed in the query is translated (in a similar way to ARES) into a boolean one using the appropriate metric and the resulting query is used to select tuples. Then, an ordering process takes place, relying on the calculation of distances (by means of associated metrics) for the elementary vague conditions. The global distance attached to a selected tuple in case of a disjunctive query is the smallest of distances related to each vague condition. For conjunctive queries, the global distance is obtained as the root of the sum of the squares (i.e., the Euclidean distance) of distances tied to each vague condition.

[9] A User Interface to Relational Databases that Permits VAGUE Queries

Note that in VAGUE, the users cannot provide their own similarity thresholds for each vague condition but when a vague query does not match any data, VAGUE doubles all searching radii simultaneously. Thus the search performance can be considerably deteriorated.

IQE

Another system that supports similarity-based search over relational databases is the IQE[10] system [NK03]. IQE converts the imprecise[11] query into equivalent precise queries that appear in an existing query workload. Such queries are then used to answer the user-given imprecise query. More precisely, given the workload, the main idea of IQE is to map the user's imprecise query q_i to a precise query q_p by tightening the operator in the query condition. For example, tightening the operator 'similar-to' ($\approx$) to 'equal-to' ($=$) in the imprecise query '`Salary` $\approx$ 40K€' gives us the precise query '`Salary` $=$ 40K€'. Then IQE computes the similarity of q_p to all queries in the workload. To estimate the similarity between two queries, IQE uses the document similarity metric (Jaccard Similarity [HGKI02]) over the answer sets of the two queries. A minimal similarity threshold τ is used to prune the number of queries similar to q_p. Finally, the answer to q_i is the union of the answers of the precise queries similar to q_p, with each tuple in the union inheriting the similarity of its generating precise query. Although IQE is a useful system, a workload containing past user queries is required, which is unavailable for new online databases.

Nearest Neighbors

In the approach known as nearest neighbors, database records and queries are viewed as points (i.e., feature vectors) in a multidimensional space S with a metric $\mathcal{M}_S$ (e.g., the Euclidean distance). Here a typical query is given by an example [Zlo75] and its result set corresponds to the set of database records which are close to it according to $\mathcal{M}_S$. For instance, in image databases, the user may pose a query asking for the images most similar to a given image. This type of query is known as nearest neighbor query [RKV95] and it has been extensively studied in the past [BBKK97, IM98, AMN+98, BGRS99, WB00, Cia00, FSAA01, FTAA01].

[10] Imprecise Query Engine

[11] The authors refer to queries that involve the similarity operator ($\approx$) as imprecise queries while those involving only the usual equality ($=$) are referred as precise queries.

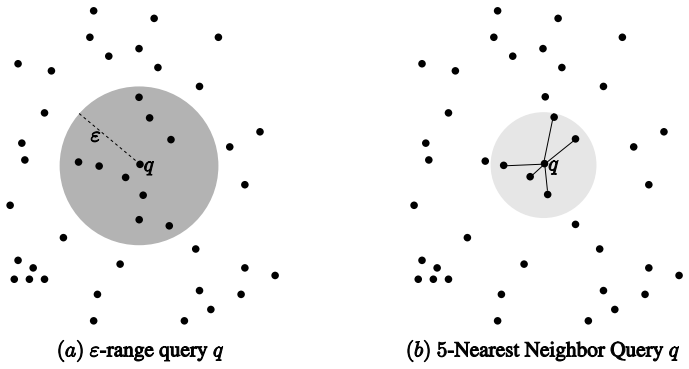
The two most important types of nearest neighbor queries (NNQ) in databases are :

- ***ε-Range Query.*** The user specifies a query object $q \in S$ and a query radius ε. The system retrieves all objects from the database $\mathrm{DB} \subset S$ that have a distance from q not exceeding ε (Figure 1.4-(a)). More formally, the result set $\mathrm{RQ}_\varepsilon^q$ is defined as follows :

$$\mathrm{RQ}_\varepsilon^q = \{t \in \mathrm{DB} | \mathcal{M}_S(q,t) \leq \varepsilon\}$$

- ***k-Nearest Neighbor Query.*** The user specifies a query object q and the cardinality k of the result set. The system retrieves the k objects from the database $\mathrm{DB} \subset S$ that have the least distance from q (Figure 1.4-(b)). More formally, the result set NN_k^q is defined as follows :

$$\forall t \in \mathrm{NN}_k^q \, , \forall t' \in \mathrm{DB} \backslash \mathrm{NN}_k^q \, , \mathcal{M}_S(q,t) < \mathcal{M}_S(q,t')$$



(a) ε-range query q (b) 5-Nearest Neighbor Query q

Figure 1.4 – Nearest neighbor query types

A naive solution for answering a given NNQ query is to scan the entire database and test for each object if it is currently among the results. Obviously, this solution is very expensive and not feasible for a very large set of objects. Several multidimensional index structures, that enable to prune large parts of the search space, were proposed. The most popular are R-Tree [Gut88] and its variants R*-tree [BKSS90], X-Tree [BKK96], SS-Tree [WJ96], etc. For a more detailed elaboration on multidimensional access methods and on the corresponding query processing techniques, we refer the interested reader to [GG98] and [BBK01].

While the approaches described in both previous subsections differ in their implementation details, their overall goal is the same — allowing the database system to return answers, related to the failed query, which is more convenient than returning nothing. In the following section, we review some works addressing the *many-answers* problem, i.e., the situation where the original query results in overabundant answers.

1.2 Handling the *Many-Answers* Problem

Due to the ever increasing amount of information stored each day into databases, user's queries often result in too many answers - many of them irrelevant to the user. This phenomenon is also commonly referred to as 'information overload', as the user expends a huge amount of effort sifting through the result set looking for interesting results.

The *many-answers* problem often stems from the specificity of the user query that is too general. A solution to reduce the number of answers consists in narrowing the query by adding new conditions. However, it is quite impossible to find conditions that can effectively restrict the retrieved data without any knowledge of the database content. To circumvent this initial issue, several techniques have been proposed, including automated ranking (Section 1.2.1) and clustering (Section 1.2.2) of query results.

1.2.1 Automated Ranking of Query Results

Automated ranking of query results is a popular aspect of the query model in Information Retrieval. It consists in providing a user with a list ranked in descending order of relevance (though the user may not have explicitly specified how) of either all query results or only the top-k subset, rather than thousands of answers ordered in a completely uninformative way. In contrast, the Boolean query model used in traditional database systems does not allow for any form of relevance ranking of the retrieved tuple set. Indeed, all tuples that satisfy the query conditions are returned without any relevance distinction between them. Thus, the result of any query is simply a partition of the database into a set of retrieved tuples, and a set of not-retrieved tuples. It is only recently that top-k queries have been introduced (see [IBS08] for a survey). Top-k queries provide the user with the k most relevant results of a given query ranked according to some scoring function. Consider a realtor database `HouseDB` with information on houses for sale, including their `Price`, `Size`, `Age`, `City`, `Zip`, `Location`, `SchoolDistrict`, `View`, `#Bedrooms`, `#Bathrooms`, `Garage` and `BoatDock`. The following query is an example of a top-k query :

SELECT	`Price` AS p, `Size` AS s, `Age` AS a
FROM	`HouseDB`
WHERE	`Zip` = 75000
ORDER BY	$f(d_{Price}(p), d_{Size}(s), d_{Age}(a))$
LIMIT	5

where LIMIT limits the number of results reported to the user, $d_A(x)$ measures the extent (score)

to which the value x of attribute A is relevant to the user and f determines how to combine the ranking according to each feature in an overall ranking. As one can observe, a fundamental requirement of top-k based approaches is that they require a scoring function that specifies which result from a large set of potential answers to a query is most relevant to the user. Achieving this requirement is generally not a straightforward endeavor, especially when users do not really know what might be useful or relevant for them.

Automatic methods of ranking answers of database (DB) queries have been recently investigated to overcome this problem, most of them being an adaptation of those employed in Information Retrieval (IR). Before discussing these methods in detail, we first review existing Information Retrieval ranking techniques. Then, we discuss related work on top-k query processing techniques in relational database systems.

1.2.1.1 IR Ranking Techniques

Automated ranking of query results has been extensively investigated in Information Retrieval [SM86, Rob97, BP98, BH98, Kle99, BRRT01, NZJ01]. Indeed, the web pages that are returned as a result to a query expressed by a set of keywords are automatically ranked so that the more 'relevant' the page is to the query, the higher it is ranked. Furthermore, among the web pages that are equally relevant, those that are more 'important' should precede the less 'important' ones. Models that have been used for this purpose include vector space and probabilistic information retrieval models which are eventually combined with link-based ranking methods to offer the user not only relevant documents but also high quality Web pages.

Vector Space Model

Vector space model [SM86] represents documents as term vectors in an N-dimensional space where N corresponds to the number of terms in the collection. Each element in a vector captures a term and its weight, defined as $w = \mathsf{tf} * \mathsf{idf}$, where tf is the term frequency in the document and idf is the inverse document frequency reflecting the general importance of the term in the entire document collection. Specifically, idf decreases the weight of terms that occur in numerous documents and, therefore, have low discriminating value. Queries are also represented as vectors of keywords. Hence, given a keyword query, the retrieved documents are ranked based on their similarity to the query. A range of measures exists to calculate this similarity ; the most common one is the cosine similarity measure [BYRN99] defined as the Cosine of the angle between the query and the document vector representations.

Probabilistic Model

In the probabilistic model [Rob97], documents and queries are also viewed as vectors, whereas the vector space similarity measure is replaced by a probabilistic matching function. More precisely, given a document collection D and a query q, probabilistic-based approaches formally rank documents by decreasing order of their odds of relevance to non-relevance using a score function defined as follows :

$$score(d) = \frac{P(Rel|d)}{P(\overline{Rel}|d)} = \frac{\frac{P(d|Rel)P(Rel)}{P(d)}}{\frac{P(d|\overline{Rel})P(\overline{Rel})}{P(d)}} \approx \frac{P(d|Rel)}{P(d|\overline{Rel})}$$

where Rel denotes the set of relevant documents, $\overline{Rel} = (D \setminus Rel)$ the set of irrelevant ones and $P(Rel|d)$ (resp., $P(\overline{Rel}|d)$) the probability of relevance (resp., non-relevance) of document d w.r.t. the query q. The higher the ratio of the probability of relevance to non-relevance is w.r.t. a document d, then the more likely document d is to be relevant to a user query q.

In the above formula, the second equality is obtained using the Bayes formula [Bay63], whereas the final simplification follows from the fact that $p(Rel)$ and $p(\overline{Rel})$ are the same for every document d and thus are mere constant values that do not influence the ranking of documents. Note that Rel and $\overline{Rel}$ are unknown at query time and consequently are estimated as accurately as possible, on the basis of whatever data has been made available to the system for this purpose. The usual techniques in Information Retrieval make some simplifying assumptions, such as estimating Rel through user feedback, approximating $\overline{Rel}$ as D (since Rel is usually small compared to D) and assuming some form of independence between query terms (e.g., the Binary Independence Model, the Linked Dependence Model, or the Tree Dependence Model [YM98, GF04]). INQUERY [CCH92] is an example of this model.

Link-based Ranking Methods

The link-based ranking methods are based on how the pages on the Internet link to each other [BP98, BH98, Kle99, BRRT01, NZJ01]. Indeed, the relevance of a page is not only decided by the page content, but is also based on the linkage among pages. An example of a ranking algorithm based on link analysis is the PageRank algorithm [BP98] introduced by Google[12]. PageRank computes Web page scores by exploiting the graph inferred from the link structure of the Web. Its underlying motivation is that pages with many backlinks are more important than pages with only a few backlinks. So basically, a page's rank in Google's search results is higher

[12] http ://www.google.com

if many, preferably important, pages link to that page. The higher the PageRank is, the more relevant the page is (according to Google). Another example of a ranking algorithm using link analysis is the HITS[13] algorithm [Kle99]. The HITS algorithm suggests that each page should have a separate 'authority' rating (based on the links going to the page) and a 'hub' rating (based on the links going from the page). The intuition behind the algorithm is that important hubs have links to important authorities and important authorities are linked by important hubs.

For more details and more general sources about Information Retrieval ranking methods, please refer to [MRS08].

1.2.1.2 DB Ranking Techniques

Automated ranking of database query results has been extensively investigated in recent years. Examples include [CD03], [CDHW04], [SWHL06], [WFSP00] and [MBKB02]. These approaches take a user's query - which typically specify simple selection conditions on a small set of attributes - and use diverse knowledge sources to automatically estimate the ranking of its results. For instance, the systems proposed in [CD03], [CDHW04] and [SWHL06] use workload and/or database statistics while [WFSP00] and [MBKB02] use relevance feedback from the user.

Chaudhuri et al's System

In [CD03], the authors proposed a ranking function (QF_W) that leverages workload information to rank the answers to a database query. It is based on the frequency of occurrence of the values of unspecified[14] attributes. For example, consider a home-buyer searching for houses in `HouseDB`. A query with a not very selective condition such as '`City` = Paris AND `#Bedrooms` = 2' may result in too many tuples in the answer, since there are many houses with two bedrooms in Paris. The proposed system uses workload information and examines attributes other than `City` and `#Bedrooms` (i.e., attributes that are not specified in the query) to rank the result set. Thus, if the workload contains many more queries for houses in Paris's 15th arrondissement (precinct) than for houses in Paris's 18th arrondissement, the system ranks two bedroom houses in the 15th arrondissement higher than two bedroom houses in the 18th arrondissement. The intuition is that if the 15th arrondissement is a wonderful location, the workload

[13] Hyperlink-Induced Topic Search

[14] In the case of information retrieval, ranking functions are often based on the frequency of occurrence of query values in documents (term frequency, or `tf`). However, in the database context, `tf` is irrelevant as tuples either contain or do not contain a query value. Hence ranking functions need to consider values of unspecified attributes.

will contain many more queries for houses in the 15th than for houses in the 18th. More formally, consider one relation R. Each tuple in R has N attributes $A_1, \ldots, A_N$. Further, let q be a user's query with some number of attributes specified (e.g., $A_1, A_2 \ldots A_i$ and $i < N$) and the rest of them unspecified (e.g., $A_{i+1}, \ldots, A_N$). The relevance score of an answer t is defined as follows :

$$\mathsf{QF}_W(t) = \sum_{k=i+1}^{N} \frac{F(t.A_k)}{F_{max}}$$

where $F(t.A_k)$ is the frequency of occurrence of value $t.A_k$ of attribute A_k in the workload W and F_{max} the frequency of the most frequently occurring value in W.

PIR

In the PIR[15] system [CDHW04], the authors adapted and applied principles of probabilistic models from Information Retrieval to structured data. Given a query, the proposed ranking function depends on two factors : (a) a global score that captures the global importance of unspecified attribute values, and (b) a conditional score that captures the strengths of dependencies (or correlations) between specified and unspecified attribute values. For example, for the query '`City` = Marseille AND `View` = Waterfront', a house with '`SchoolDistrict` = Excellent' gets a high rank because good school districts are globally desirable. A house with also '`BoatDock` = Yes' gets a high rank because people desiring a waterfront are likely to want a boat dock. These scores are estimated using past workloads as well as data analysis, e.g., past workload may reveal that a large fraction of users seeking houses with a waterfront view have also requested boat docks. More precisely, under the same notations and hypotheses that we have used for the previous approach, the relevance score of a tuple t is computed as follows :

$$\mathsf{PIR}(t) = \frac{P(Rel|t)}{P(\overline{Rel}|t)} \approx \prod_{k=i+1}^{N} \frac{P(t.A_k|W)}{P(t.A_k|R)} * \prod_{k=i+1}^{N} \prod_{l=1}^{i} \frac{P(t.A_l|t.A_k, W)}{P(t.A_l|t.A_k, R)}$$

where the quantities $P(t.A_k|W)$ and $P(t.A_k|R)$ are simply the relative frequencies of each distinct value $t.A_k$ respectively in the workload W and in the relation R, while the quantities $P(t.A_l|t.A_k, W)$ and $P(t.A_l|t.A_k, R)$ are estimated by computing the confidences of pair-wise association rules [AMS+96] in W and R, respectively. Note that the score in the above formula is composed of two large factors. The first factor is the global part of the score, while the second one is the conditional part of the score. This approach is implemented in STAR [KDH+07].

[15] Probabilistic Information Retrieval

Note that in both Chaudhuri's system [CD03] and the PIR system [CDHW04], the atomic quantities $F(x)$, $P(x|W)$, $P(x|R)$, $P(y|x,W)$ and $P(y|x,R)$ are pre-computed and stored in special auxiliary tables for all distinct values x and y in the workload and the database. Then at query time, both approaches first select the tuples that satisfy the query condition, then scan and compute the score for each such tuple using the information in the auxiliary tables, and finally returns the top-k tuples. The main drawback of both [CD03] and [CDHW04] is their high storage and maintenance costs of auxiliary tables. Moreover, they require a workload containing past user queries as input, which is not always available (e.g., new online databases).

QRRE

In the QRRE[16] system [SWHL06], the authors proposed an automatic ranking method, which can rank the query results from an E-commerce Web database R using only data analysis techniques. Consider a tuple $t = \langle t.A_1, \ldots, t.A_N \rangle$ in the result set T_q of a query q that is submitted by a buyer. QRRE assigns a weight w_i to each attribute A_i that reflects its importance to the user. w_i is evaluated by the difference (e.g., The Kullback-Leibler divergence [DHS00]) between the distribution (histogram) of A_i's values over the result set T_q and their distribution (histogram) over the whole database R. The bigger the divergence, the more A_i is important for a buyer. For instance, suppose the database `HouseDB` contains houses for sale in France and consider the query q with the condition '`View` = Waterfront'. Intuitively, the `Price` values of the tuples in the result set T_q distribute in a small and dense range with a relatively high average, while the `Price` values of tuples in `HouseDB` distribute in a large range with a relatively low average. The distribution difference shows a close correlation between the unspecified attribute, namely, `Price`, and the query '`View` = Waterfront'. In contrast, attribute `Size` is less important for the user since its distribution in houses with a waterfront view may be similar to its distribution in the entire database `HouseDB`. Besides the attribute weight, QRRE also assigns a preference score p_i to each attribute value $t.A_i$. p_i is computed based on the following two assumptions :

1. a product with a lower price is always more desired by buyers than a product with a higher price if the other attributes of the two products have the same values. For example, between two houses that differ only in their price, the cheapest one is preferred. Hence, QRRE assigns a small preference score to a high `Price` value and a large preference score to a low `Price` value ;

[16] Query Result Ranking for E-commerce

2. a non-Price attribute value with higher 'desirableness' for the user corresponds to a higher price. For example, a large house, which most buyers prefer, is usually more expensive than a small one. Thus, in the case of a non-Price attribute A_i, QRRE first converts its value $t.A_i$ to a `Price` value pv which is the average price of the products for $A_i = t.A_i$ in the database R. Then, QRRE assigns a large preference score to $t.A_i$ if pv is large.

Finally, the attribute weight and the value preference score are combined to calculate the ranking score for each tuple $t \in T_q$, as follows :

$$\mathsf{QRRE}(t) = \sum_{i=1}^{n} w_i p_i$$

The tuples' ranking scores are sorted and the top-K tuples with the largest ranking scores are presented to the user first. QRRE is a useful automated ranking approach for the *many-answers* problem. It does not depend on domains nor require workloads. However, this approach may imply high response times, especially in the case of low selectivity queries, since different histograms need to be constructed over the result set (i.e., at query time).

Feedback-based Systems

Another approach to rank query's results, which is different from those discussed above, is to prompt the user for feedback on retrieval results and then use this feedback on subsequent retrievals to effectively infer which tuples in the database are of interest to the user. Relevance Feedback techniques were studied extensively in the context of image retrieval [WFSP00, MBKB02] and were usually paired with the query-by-example approach [Zlo75]. The basic procedure of these approaches is as follows :

1. the user issues a query ;
2. the system returns an initial set of results ;
3. the user marks some returned tuples as relevant or non-relevant ;
4. the system constructs a new query that is supposed to be close to the relevant results and far from those which are non-relevant ; and
5. the system displays the results that are most similar to the new query.

This procedure can be conducted iteratively until the user is satisfied with the query results. Relevance feedback-based approaches provide an effective method for reducing the number of query results. However, they are not necessarily popular with users. Indeed, users are often reluctant to provide explicit feedback, or simply do not wish to prolong the search interaction.

Furthermore, it is often harder to understand why a particular tuple is retrieved after the relevance feedback algorithm has been applied.

Once the scoring function is defined, the DB ranking techniques discussed in this subsection adapt and use available top-k query processing algorithms [IBS08] in order to quickly provide the user with the k most relevant results of a given query. In the following subsection, we briefly review top-k query processing methods in relational database systems.

1.2.1.3 Efficient Top-k Query Processing

Assume a relation R with attributes $A_1, \ldots, A_N$ and a query q over R. Further, one supposes that each tuple $t = \langle t.A_1, \ldots, t.A_N \rangle$, in the result set T_q of q, has N attribute-oriented scores $s_1, \ldots, s_N$. Each s_i measures the extent (score) to which the value $t.A_i$ of tuple t on attribute A_i is relevant to the user. For the top-k problem, T_q could alternatively be seen as a set of N sorted lists L_i of $|T_q|$ (the number of tuples in T_q) pairs (t, s_i), $t \in T_q$. Hence, for each attribute A_i, there is a sorted list L_i in which all $|T_q|$ results are ranked in descendant order. Entries in the lists could be accessed randomly from the tuple identifier or sequentially from the sorted score. The main issue for top-k query processing is then to obtain the k tuples with the highest overall scores computed according to a given aggregation function $\mathsf{agg}(s_1, \ldots, s_N)$ of the attribute scores s_i. The aggregation function agg used to combine ranking criteria has to be monotone ; that is, agg must satisfy the following property :

$$\mathsf{agg}(s_1, \ldots, s_N) \leq \mathsf{agg}(s'_1, \ldots, s'_N) \text{ if } s_i \leq s'_i \text{ for every } i$$

The naive algorithm consists in looking at every entry (t, s_i) in each of the sorted lists L_i, computing the overall grade of every object t, and returning the top k answers. Obviously, this approach is unnecessarily expensive as it does not take advantage of the fact that only the k best answers are part of the query answer and the remaining answers do not need to be processed.

Several query answering algorithms have been proposed in the literature to efficiently process top-k queries. The most popular is the Threshold Algorithm (TA) independently proposed by several groups [FLN01, NR99, GBK00].

The TA algorithm works as follows :

1. do sorted access in parallel to each of the N sorted lists. As a tuple t is seen under sorted access in some list, do random access to the other lists to find the score of t in every list and compute the overall score of t. Maintain in a set TOP the k seen tuples whose overall scores are the highest among all tuples seen so far ;

2. for each list L_i, let s_i^* be the last score seen under sorted access in L_i. Define the threshold to be $\tau = \mathsf{agg}(s_1^*, \ldots, s_N^*)$. If TOP involves k tuples whose overall scores are higher than or equal to τ, then stop doing sorted access to the lists. Otherwise, go to step 1 ;
3. return TOP.

Table 1.2 – The Threshold Algorithm - an example with 3 lists

Position	L_1	L_2	L_3
1	$(t_5, 21)$	$(t_4, 34)$	$(t_3, 30)$
2	$(t_2, 17)$	$(t_1, 29)$	$(t_4, 14)$
3	$(t_4, 11)$	$(t_0, 29)$	$(t_0, 9)$
4	$(t_3, 11)$	$(t_3, 26)$	$(t_5, 7)$
5	$(t_6, 10)$	$(t_5, 9)$	$(t_2, 1)$
6	$(t_7, 10)$	$(t_9, 7)$	$(t_8, 1)$

Table 1.2 shows an example with three Lists L_1, L_2 and L_3. Assume that the top-k query requests the top-2 tuples and the aggregation function agg is the summation function SUM. TA first scans the first tuples in all lists which are t_5, t_4, and t_3. Hence the threshold value at this time is $\tau = 21 + 34 + 30 = 85$. Then TA calculates the aggregated score for each tuple seen so far by random accesses to the three lists. We get the aggregated score for t_5 $\mathsf{SUM}(t_5) = 21+9+7 = 37$, for t_4 $\mathsf{SUM}(t_4) = 11+34+14 = 59$ and for t_3 $\mathsf{SUM}(t_3) = 11+26+30 = 67$. TA maintains the top-2 tuples seen so far which are t_3 and t_4. As neither of them has an aggregated score greater than the current threshold value $\tau = 85$, TA continues to scan the tuples at the second positions of all lists. At this time, the threshold value is recomputed as $\tau = 17 + 29 + 14 = 60$. The new tuples seen are t_1 and t_2. Their aggregated scores are retrieved and calculated as $\mathsf{SUM}(t_1) = 0 + 29 + 0 = 29$ and $\mathsf{SUM}(t_2) = 17 + 0 + 1 = 18$. TA still keeps tuples t_3 and t_4 since their aggregated scores are higher than those of both t_1 and t_2. Since only t_3 has an aggregated score greater than the current threshold value $\tau = 60$, TA algorithm continues to scan the tuples in the third positions. Now the threshold value is $\tau = 11 + 29 + 9 = 49$ and the new tuple seen is t_0. TA computes the aggregated score for t_0 which is 38. t_3 and t_4 still maintain the two highest aggregated scores which are now greater than the current threshold value $\tau = 49$. Thereby, TA terminates at this point and returns t_3 and t_4 as the top-2 tuples. Note that, in this example, TA avoids accessing the tuples t_6, t_7, t_8 and t_9.

For more details about top-k processing techniques in relational databases, we refer the interested reader to [IBS08].

1.2.2 Clustering of Query Results

Clustering of query results is the operation of grouping the set of answers into meaningful clusters (or groups). This allows the user to select and browse clusters that most closely match what she/he is looking for while ignoring the irrelevant ones. This idea has been traditionally used for organizing the results of Web search engines [JR71, vRC75, Cro80, Voo85, HP96, JMF99, ZE99, ZHC+04, CKVW06, FG05] but has only recently been adapted in the context of relational database [CCH04, BRM05, LWL+07]. In the following, we review existing search results clustering techniques in both information retrieval (IR) and relational database (DB) systems.

1.2.2.1 IR clustering techniques

Document clustering has been used in information retrieval for many years. Originally, it aimed at improving search efficiency by reducing the number of documents that needed to be compared to the query. The rationale was that by partitioning the document collection in clusters, an information retrieval system could restrict the search to only some of them. It was only with the work of Jardine and Van Rijsbergen that clustering became associated with search effectiveness [JR71].

The motivation for the use of clustering as a way to improve retrieval effectiveness lies in the *cluster hypothesis*. The cluster hypothesis, as proposed by Jardine and van Rijsbergen, states that the association between documents conveys information about the "relevance of documents to the request" [JR71]. In other words, if a document is relevant to an *information need* expressed in a query[17], then similar documents are also likely to be relevant to the same information need. So, if similar documents are grouped into clusters, then one of these clusters contains the relevant documents (or most of them). Therefore, finding this cluster could improve search effectiveness.

There are various ways in which the cluster hypothesis could be exploited. One way is to implement cluster-based retrieval. In cluster-based retrieval, the strategy is to build a clustering of the entire collection in advance and then retrieve clusters based on how well their representations (e.g., a set of keywords) match the upcoming query. A hierarchical clustering technique is typically used in these approaches, and different strategies for matching the query against the document hierarchy have been proposed, most notably a top-down or a bottom-up search

[17] In information retrieval systems, the information need is what the user desires to know from the stored data, to satisfy some intended objective (e.g., data analysis, decision making). However, the query is what the user submits to the system in an attempt to fulfill that information need.

and their variants [JR71, vRC75, Cro80, Voo85]. Similarly, search engines (e.g., Yahoo) and product catalog search (e.g., eBay) use a category structure created in advance and then group search results into separate categories. In all these approaches, if a query does not match any cluster representation of one of the pre-defined clusters or categories, then it fails to match any documents even if the document collection contains relevant results. It is worth noticing that this problem is not intrinsic to clustering, but is due to the fact that keyword representation of clusters is often insufficient to apprehend the meaning of documents in a cluster.

An alternative way of using the cluster hypothesis is in the presentation of retrieval results, that is by presenting, in a clustered form, only documents that have been retrieved in response to the query. This idea was first introduced in the Scatter/Gather system [HP96] which is based on a variant of the classical k-means algorithm [HW79]. Since then, several classes of algorithms have been proposed such as STC [ZE99], SHOC [ZHC+04], EigenCluster [CKVW06], SnakeT [FG05]. Note that such algorithms introduce a noticeable time overhead to the query processing, due to the large number of results returned by the search engine. The reader is referred to [MRS08] for more details on IR clustering techniques.

All the above approaches testify that there is a significant potential benefit in providing additional structure in large answer sets.

1.2.2.2 DB clustering techniques

The SQL group-by operator allows grouping query results. It classifies the results of a query into groups based on a user-selected subset of fields. However, it partitions the space only by identical values. For instance, if there are 1000 different zip-codes, 'group-by `Zip`' returns 1000 groups. In the last few years, traditional clustering techniques have been employed by the database research community to overcome this shortcoming [CCH04, BRM05, LWL+07].

Chakrabarti et al's System

In [CCH04], the authors proposed an automatic method for categorizing query results. This method dynamically creates a navigation tree (i.e., a hierarchical category structure) for each query q based on the contents of the tuples in the answer set T_q. A hierarchical categorization of T_q is a recursive partitioning of the tuples in T_q based on the data attributes and their values, i.e., the query results are grouped into nested categories. Figure 1.5 shows an example of a hierarchical categorization of the results of the query '`City` = Paris AND `Price` $\in [150, 300]$K€'. At each level, the partitioning is done based on a single attribute in the result set T_q, and this

attribute is the same for all nodes at that level. Furthermore, once an attribute is used as a categorizing attribute at any level, it is not repeated at a later level. For example, `Price` is the categorizing attribute of all nodes at Level 2. The partitions are assigned descriptive labels and form a categorization of the result set based on that attribute. For example, the first child of the root in Figure 1.5 has label '`Location` = 18th arrondissement' while its own first child has label '`Price` $\in [200, 225]$K€'.

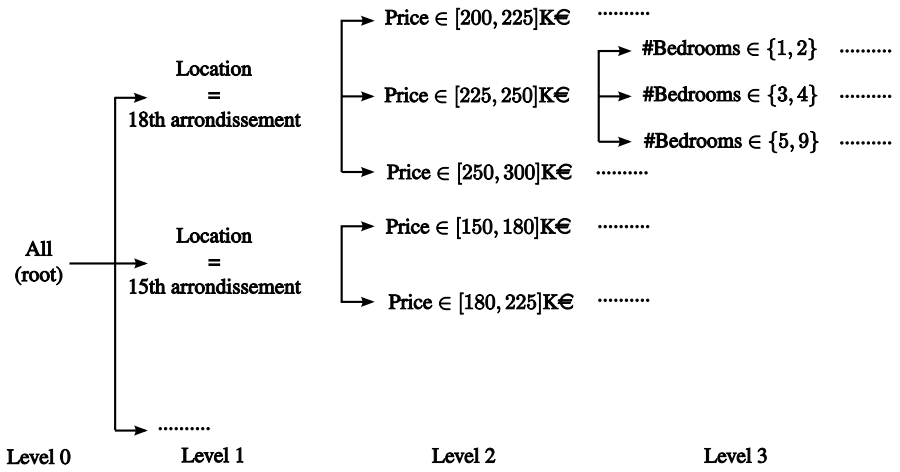


Figure 1.5 – Example of hierarchical categorization of query results

The order in which the attributes appear in the tree, and the values used to split the domain of any attribute are inferred by analyzing the aggregate knowledge of previous user behaviors — using the workload. Indeed, the attributes that appear most frequently in the workload are presented to the user earlier (i.e., at the highest levels of the tree). The intuition behind this approach is that the presence of a selection condition on an attribute in a workload reflects the user's interest in that attribute. Furthermore, for each attribute A_i, one of the following two methods is used to partition the set of tuples $tset(C)$ contained in a category C depending on whether A_i is categorical or numeric :

- If A_i is a categorical attribute with discrete values $\{v_1, \ldots, v_k\}$, the proposed algorithm simply partitions $tset(C)$ into k categories, one category C_j corresponding to a value v_j. Then, it presents them in the decreasing order of $\mathsf{occ}(A_i = v_j)$, i.e., the number of queries in the workload whose selection condition on A_i overlaps with $A_i = v_j$;
- Otherwise, assume the domain of attribute A_i is the interval $[v_{min}, v_{max}]$. If a significant number of query ranges (corresponding to the selection condition on A_i) in the workload begins or ends at $v \in [v_{min}, v_{max}]$, then v is considered as a good point to split

$[v_{min}, v_{max}]$. The intuition here is that most users would be interested in just one bucket, i.e., either in the bucket $A_i \leq v$ or in the bucket $A_i > v$ but not both.

This approach provides the user with navigational facilities to browse query results. However, it requires a workload containing past user queries as input, which is not always available. Furthermore, the hierarchical category structure is built at query time, and hence the user has to wait a long time before the results can be displayed.

OSQR

In [BRM05], the authors proposed OSQR[18], an approach for clustering database query results based on the agglomerative single-link approach [JMF99]. Given an SQL query as input, OSQR explores its result set, and identifies a set of terms (called the query's context) that are the most relevant to the query ; each term in this set is also associated with a score quantifying its relevance. Next, OSQR exploits the term scores and the association of the rows in the query result with the respective terms to define a similarity measure between the terms. This similarity measure is then used to group multiple terms together ; this grouping, in turn, induces a clustering of the query result rows. More precisely, consider a query q on a table R, and let T_q denote the result of the query q. OSQR works as follows :

1. scan T_q and assign a score s_x to each attribute value x (or term) in T_q. The terms' scores (similar to $\mathsf{tf} * \mathsf{idf}$ scores used in information retrieval) are defined in such a way that higher scores indicate attributes values that are popular in the query result T_q and are rare in $R \setminus T_q$;
2. compute the context of q as the set of terms $q_{context}$ with scores exceeding a certain threshold (a system parameter) ;
3. associate to each term x in $q_{context}$ the cluster C_x, i.e., the set of tuples of T_q in which the attribute value x appears. The tuples in T_q that are not associated with any term $x \in q_{context}$ are termed 'outliers'. These rows are not processed any further ;
4. iteratively merge the two most similar clusters until a stopping condition is met. The similarity $\mathsf{sim}(C_x, C_y)$ between each pair of clusters C_x and C_y ($x, y \in q_{context}$) is defined as follows :
$$\mathsf{sim}(C_x, C_y) = \frac{s_y|C_x - C_y| + s_x|C_y - C_x|}{s_y|C_x| + s_x|C_y|}$$
where $|.|$ denotes the cardinality of a set.

[18] Overlapping cluStering of Query Results

OSQR's output is a dendrogram that can be browsed from its root node to its leaves, where each leaf represents a single term x in $q_{context}$ and its associated tuple set C_x.

The above approach has many desirable features : it generates overlapping clusters, associates a descriptive 'context' with each generated cluster, and does not require the query workload. However, note that some query results (tuples that are not associated with any term in $q_{context}$) are ignored and therefore not included in the output result. Moreover, this approach may imply high response times, especially in the case of low selectivity queries, since both scoring and clustering of terms are done on the fly.

Li et al's System

In a recent work [LWL+07], the authors generalized the SQL *group-by* operator to enable grouping (based on the proximity of attribute values) of database query results. Consider a relation R with attributes $A_1, \ldots, A_N$ and a user's query q over R with a *group-by* clause on a subset X of R's numeric attributes. The proposed algorithm first divides the domain of each attribute $A_i \in X$ into p_i disjoint intervals (or bins) to form a grid of $\prod_{1 \leq i \leq |X|} p_i$ buckets (or cells). Next, this approach identifies the set of buckets $C = \{b_1, \ldots, b_m\}$ that holds the results of q, and associates to each bucket b_i a virtual point v_i, located at the center of that bucket. Finally, a k-means algorithm [HW79] is performed on these virtual points (i.e., $\{v_1, \ldots, v_m\}$) to obtain exactly k clusters of q's results. The k parameter is given by the end-user. For example, consider a user's query that returns 10 tuples $t_1, \ldots, t_{10}$ and the user needs to partition these tuples into 2 clusters, using two attributes A_1 and A_2. Figure 1.6 shows an example of a grid over $t_1, \ldots, t_{10}$ by partitioning attributes A_1 and A_2. The bins on A_1 and A_2 are $\{[0,3), [3,6), [6,9)\}$ and $\{[0,10), [10,20), [20,30)\}$, respectively. The two clusters C_1 (i.e. $A_1 \in [0,3] \wedge A_2 \in [10,30]$) and C_2 (i.e., $A_1 \in [6,9] \wedge A_2 \in [0,10]$) are returned to that user. C_1 contains 6 tuples t_1, t_3, t_4, t_6, t_9 and t_{10}, whereas C_2 contains 4 tuples t_2, t_5, t_7 and t_8.

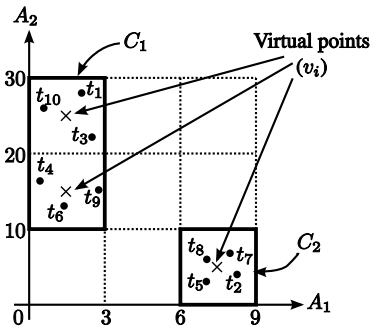


Figure 1.6 – Example of a grid in two dimensions

This approach is efficient. Indeed, it relies on a bucket-level clustering, which is much more efficient than the tuple-level one, since the number of buckets is much smaller than the number of tuples. However, the proposed algorithm requires the user to specify the number of clusters k, which is difficult to know in advance, but has a crucial impact on the clustering result. Further, this approach generates flat clustering of query results and some clusters may contain a very large number of results, although, this is exactly the kind of outcome this technique should avoid.

1.3 Flexible/User-Friendly Database Querying

A typical problem with traditional database query languages like SQL is a lack of flexibility. Indeed, they are plagued by a fundamental problem of specificity (as we have seen in Sections 1.1 and 1.2) : if the query is too specific (with respect to the dataset), the response is empty ; if the query is too general, the response is an avalanche. Hence, it is difficult to cast a query, balanced on this scale of specificity, that returns a reasonable number of results. Furthermore, they expect users to know the schema of the database they wish to access.

Recently, many flexible/user-friendly querying techniques have been proposed to overcome this problem. The main objective of these techniques is to provide human-oriented interfaces which allow for a more intelligent and human-consistent information retrieval and hence, diminish the risk of both empty and many answers. Examples include preference queries (Section 1.3.1), fuzzy queries (Section 1.3.2) and keyword search (Section 1.3.3).

1.3.1 Preference Queries

The first way of introducing flexibility within the query processing is to cope with user preferences. More precisely, the idea is to select database records with Boolean conditions ('hard' constraints) and then to use preferences ('soft' constraints) to order the previously selected records.

Lacroix and Lavency [LL87] were the first to introduce the notion of a preference query to the database field. They proposed an extension of the relational calculus in which preferences for tuples satisfying given logical conditions can be expressed. For instance, one could say : pick the tuples of R satisfying $Q \wedge P_1 \wedge P_2$; if the result is empty, pick the tuples satisfying $Q \wedge P_1 \wedge \neg P_2$; if the result is empty, pick the tuples satisfying $Q \wedge \neg P_1 \wedge P_2$. In other words, Q is a 'hard' constraint, whereas P_1 and P_2 are preferences or 'soft' constraints. Since then extensive

investigation has been conducted, and two main types of approaches have been distinguished in the literature to deal with the user's preferences, namely, quantitative and qualitative[Cho03].

1.3.1.1 Quantitative Preferences

The quantitative approach expresses preferences using scoring functions, which associate a numeric score with every tuple of the query, e.g. "I like tuple t with score 0.5". Then tuple t_1 is preferred over tuple t_2 if and only if the score of t_1 is higher than the score of t_2. Agrawal et al. [AW00] provided a framework for expressing and combining such kinds of preference functions.

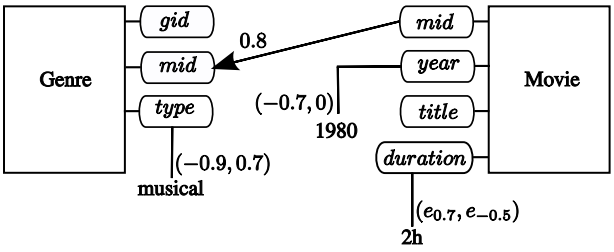


Figure 1.7 – An example of a personalization graph, indicating among others a preference for movies of duration around 2h and a big concern about the genre of a movie. The first value between brackets indicates the preference for the presence of the associated value, the second indicates the preference for the absence of the associated value. The function e stands for an elastic preference. (cf. [KI05])

Recently, Koutrika et al. [KI05] presented a richer preference model which can associate degrees of interest (like scores) with preferences over a database schema. Thus, from this aspect, it seems to follow a quantitative approach. These preferences are all kept in a user profile. A user profile is viewed as a directed graph $\mathcal{G}\langle\mathcal{V},\mathcal{E}\rangle$ (called Personalization Graph), where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ the set of edges. Nodes in $\mathcal{V}$ are (a) relation nodes, one for each relation in the schema, (b) attribute nodes, one for each attribute of each relation in the schema, and (c) value nodes, one for each value that is of any interest to a particular user. Edges in $\mathcal{E}$ are (a) selection edges, representing a possible selection condition from an attribute to a value node, and (b) join edges, representing a join between attribute nodes. An example of a user profile is given in Figure 1.7. Furthermore, the authors proposed a query personalization algorithm which exploits the user profile to dynamically enrich (personalize) user queries prior to their execution. For example, the personalization of a user query can add selection conditions to a query, meaning that the user obtains a subset of the answer to the initial query. In general, it is expected that this

subset contains the most interesting tuples (with respect to the user preferences) of the global answer.

Koutrika et al's approach [KI05] provides one of the first solutions towards modelling of user preferences in Database systems in the form of a structured profile. In [KBL07], the authors elaborated a taxonomy of the most important knowledge composing a user profile and proposed a generic model that can be instantiated and adapted to each specific application. [KBL07] also extended the query personalization algorithm proposed in [KI05] to a context where the queried data are stored in different databases located on distant sites.

1.3.1.2 Qualitative Preferences

The qualitative approach intends to directly specify preferences between the tuples in the query answer, typically using binary preference relations, e.g., "I prefer tuple t_1 to tuple t_2". These kinds of preference relations can be embedded into relational query languages through relational operators or special preference constructors, which select from their input the set of the most preferred tuples. This approach is, among others, taken by Chomicki [Cho02, Cho03] using the winnow operator $\succ$ and Kießling [Kie02] in his PreferenceSQL best match only model.

To get an idea of the representation for this approach, consider the following preference from [Cho03], which specifies a preference of white wine over red when fish is served, and red wine over white, when meat is served, over a relation `MealDB` with attributes `Dish` (d), `DishType` (dt), `Wine` (w), `WineType` (wt) :

$$\begin{aligned}(d, dt, w, wt) \succ (d', dt', w', wt') \quad \equiv \quad & (d = d' \wedge dt = \text{fish} \wedge wt = \text{white} \\ & \wedge\, dt' = \text{fish} \wedge wt' = \text{red}) \\ & \vee\, (d = d' \wedge dt = \text{meat} \wedge wt = \text{red} \\ & \wedge\, dt' = \text{meat} \wedge wt' = \text{white})\end{aligned}$$

Another example representation of a qualitative preference, over a relation `CarDB` with attributes `Make`, `Year`, `Price` and `Miles`, is the following preference for cheap cars manufactured by Benz, and prior to 2005 but not before 2003, using PreferenceSQL from [Kie02] :

SELECT	*
FROM	`CarDB`
WHERE	`Make` = Benz
PREFERRING	(LOWEST(`Price`) AND `Year` BETWEEN $2003, 2005$)

Note that the qualitative approach is more general than the quantitative one, since one can define preference relations in terms of scoring functions, whereas not every preference relation can be captured by scoring functions. For example, consider the relation `BookDB(ISBN, Vendor, Price)` and its instance shown in Table 1.3. The preference "if the same ISBN, prefer lower price to higher price" gives the preferences "b_2 to b_1" and "b_1 to b_3". There is no preference between the first three books (i.e., b_1, b_2 and b_3) and the fourth one (i.e., b_4). Thus, the score of the fourth tuple should be equal to all of the scores of the first three tuples. But this implies that the scores of the first three tuples are the same, which is not possible since the second tuple is preferred to the first one which in turn is preferred to the third one.

Table 1.3 – Excerpt of the `BookDB` relation

ID	ISBN	Vendor	Price
b_1	0679726691	BooksForLess	14.75
b_2	0679726691	LowestPrices	13.50
b_3	0679726691	QualityBooks	18.80
b_4	0062059041	BooksForLess	7.30

1.3.2 Fuzzy Queries

The second and most popular approach of flexible query processing advocates the use of the fuzzy sets theory. More precisely, the idea is to allow end-users to formulate database queries using fuzzy terms that best capture their perception of the domain and then to use them to filter and rank relevant data.

1.3.2.1 Fuzzy Sets Theory

Fuzzy set theory, introduced by Zadeh [Zad56, Zad75, Zad99], is a mathematical tool for translating user's perception of the domain (often formulated in a natural language) into computable entities. Such entities are called fuzzy terms (linguistic labels). Fuzzy terms are represented as fuzzy sets and may be fuzzy values (e.g., *young*), fuzzy comparison operators (e.g., *much greater than*), fuzzy modifiers (e.g., *very*, *really*) or fuzzy quantifiers (e.g., *most*).

Mathematically, a fuzzy set F of a universe of discourse[19] U is characterized by a membership function μ_F given by :

$$\begin{array}{rccc} \mu_F : & U & \longrightarrow & [0,1] \\ & u & \longrightarrow & \mu_F(u) \end{array}$$

where $\mu_F(u)$, for each $u \in U$, denotes the degree of membership of u in the fuzzy set F. An element $u \in U$ is said to be in the fuzzy set F if and only if $\mu_F(u) > 0$ and to be a full member if and only if $\mu_F(u) = 1$. We call *support* and *kernel* of the fuzzy set F respectively the sets :

$$\textit{support}(F) = \{u_i \in U \mid \mu_F(u_i) > 0\} \text{ and } \textit{kernel}(F) = \{u_i \in U \mid \mu_F(u_i) = 1\}.$$

Furthermore, if m fuzzy sets $F_1, F_2, \ldots$ and F_m are defined over U such that $\forall i = [1..m]$,- $F_i \neq \emptyset$, $F_i \neq U$, and $\forall u \in U$, $\sum_{i=1}^{m} \mu_{F_i}(u) = 1$, the set $\{F_1, \ldots, F_m\}$ is called a fuzzy partition [Rus69] of U.

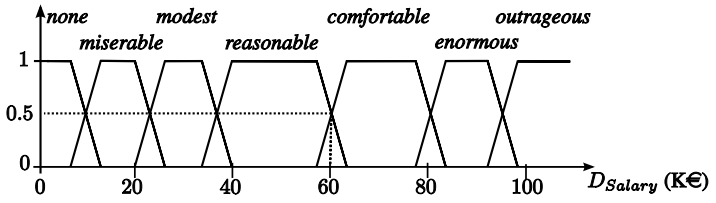


Figure 1.8 – An example of fuzzy partition defined for the attribute Salary

For example, consider the attribute `Salary` with domain $D_{Salary} = [0, 110]$K€ . A typical fuzzy partition of the universe of discourse D_{Salary} (i.e., the employees' salaries) is shown in Figure 1.8, where the fuzzy sets (values) *none*, *miserable*, *modest*, *reasonable*, *comfortable*, *enormous* and *outrageous* are defined. Here, the crisp value 60K€ has a grade of membership of 0.5 for both the *reasonable* and the *comfortable* fuzzy sets, i.e., $\mu_{reasonable}(60\text{K€}) = \mu_{comfortable}(60\text{K€}) = 0.5$.

1.3.2.2 Practical Extensions of SQL

In the literature, several extensions of SQL have been proposed to allow the use of fuzzy terms in database queries. Examples include the work of Tahani [Tah77], FQUERY [ZK96] and SQLf [BP92, BP95, BP97].

[19] Fuzzy sets can be defined in either discrete or continuous universes.

Tahani's Approach

Tahani [Tah77] was the first to propose a formal approach and architecture to deal with simple fuzzy queries for crisp relational databases. More specifically, the author proposed to use in the query condition fuzzy values instead of crisp ones. An example of a fuzzy query would be "get employees who are *young* and have a *reasonable* salary". This query contains two fuzzy predicates '`Age` = *young*' and '`Salary` = *reasonable*', where *young* and *reasonable* are words in natural language that express or identify a fuzzy set (Figure 1.9).

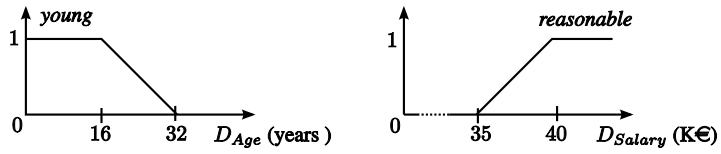


Figure 1.9 – The fuzzy sets (values) *young* and *reasonable*

Tahani's approach takes a relation R and a fuzzy query q over R as inputs and produces a fuzzy relation R_q, that is an ordinary relation in which each tuple t is associated with a matching degree γ_q within $[0, 1]$ interval. The value γ_q indicates the extent to which tuple t satisfies the fuzzy predicates involved in the query q. The matching degree, γ_q, for each particular tuple t is calculated as follows. For a tuple t and a fuzzy query q with a simple fuzzy predicate $A = l$, where A is an attribute and l is a fuzzy set defined on the attribute domain of A, $\gamma_{A=l}$ is defined as follows :

$$\gamma_{A=l}(t) = \mu_l(t.A)$$

where $t.A$ is the value of tuple t on attribute A and μ_l is the membership function of the fuzzy set l.

For instance, consider the relation `EmpDB` in Table 1.4. The fuzzy relation corresponding to the fuzzy predicate '`Age` = *young*' (resp., '`Salary` = *reasonable*') is shown in Table 1.5-(a) (resp., Table 1.5-(b)). Note that when $\gamma_q(t) = 0$, the tuple t does not belong to the fuzzy relation R_q any longer (for instance, tuple #1 in Table 1.5-(a)).

The matching function γ for a complex fuzzy query with multiple fuzzy predicates is obtained by applying the semantics of the fuzzy logical connectives, that are :

$$\gamma_{p_1 \wedge p_2}(t) = \min(\gamma_{p_1}(t), \gamma_{p_2}(t))$$
$$\gamma_{p_1 \vee p_2}(t) = \max(\gamma_{p_1}(t), \gamma_{p_2}(t))$$
$$\gamma_{\neg p_1}(t) = 1 - \gamma_{p_1}(t)$$

where p_1, p_2 are fuzzy predicates.

Table 1.4 – Excerpt of the `EmpDB` relation

Id	Salary	Age	...
1	45000	62	
2	38750	24	
3	37500	28	

Table 1.5 – Fuzzy relations : (a) $\texttt{EmpDB}_{\texttt{Age}=\textit{young}}$ (b) $\texttt{EmpDB}_{\texttt{Salary}=\textit{reasonable}}$

Id	Salary	Age	...	$\gamma_{Age=young}$
1	45000	62		0.0
2	38750	24		0.5
3	37500	28		0.25

(a)

Id	Salary	Age	...	$\gamma_{Salary=reasonable}$
1	45000	62		1
2	38750	24		0.75
3	37500	28		0.25

(b)

Table 1.6 shows the fuzzy relation corresponding to the query '`Age` = *young* AND `Salary` = *reasonable*'. Note that the $\min$ and $\max$ operators may be replaced by any t-norm and t-conorm operators [KMP00] to model the conjunction and disjunction connectives, respectively.

Table 1.6 – The fuzzy relation $\texttt{EmpDB}_{(\texttt{Age}=\textit{young}\ \text{AND}\ \texttt{Salary}=\textit{reasonable})}$

Id	Salary	Age	...	$\gamma_{(Age=young\ \wedge\ Salary=reasonable)}$
1	45000	62		0.0
2	38750	24		0.5
3	37500	28		0.25

FQUERY

In [ZK96], the authors proposed FQUERY, an extension of the Microsoft Access SQL language with capability to manipulate fuzzy terms. More specifically, they proposed to take into account the following types of fuzzy terms : fuzzy values, fuzzy comparison operators, and fuzzy quantifiers. Given a query involving fuzzy terms, the matching degree of relevant answers is calculated according to the semantics of fuzzy terms [Zad99]. In addition to the syntax

and semantics of the extended SQL, the authors have also proposed a scheme for the elicitation and manipulation of fuzzy terms to be used in queries. FQUERY has been one of the first implementations demonstrating the usefulness of fuzzy querying features for a traditional database.

SQLf

In contrast to both Tahani's approach and FQUERY which concentrated on the fuzzification of conditions appearing in the 'WHERE' clause of the SQL's SELECT statement, the query language SQLf [BP92, BP95, BP97] allows the introduction of fuzzy terms into SQL wherever they make sense. Indeed, all the operations of the relational algebra (implicitly or explicitly used in SQL's SELECT instruction) are redefined in such a way that the equivalences that occur in the crisp SQL are preserved. Thus, the projection, selection, join, union, intersection, cartesian product and set difference operations are considered. Special attention is also paid to the division operation which may be interpreted in a different way due to many possible versions of the implication available in fuzzy logic [BPR05, BPR07]. Other operations typical for SQL are also redefined, including the 'GROUP BY' clause , the 'HAVING' clause and the operators 'IN' and 'NOT IN' used along with sub-queries. A query in SQLf language has the following syntax :

SELECT	$[n\|t\|n,t]$ set of attributes
FROM	set of relations
WHERE	set of fuzzy predicates

where the parameters n and t of the select block limit the number of the answers by using a quantitative condition (the best n answers) or a qualitative condition (ones which satisfy the fuzzy predicates according to a degree higher than t).

For more details and more general sources about fuzzy querying, please refer to [Gal08].

1.3.3 Keyword Search

Keyword-searchable database systems offer a simple keyword-based search interface where users need to neither understand the underlying database schemas and structures in advance nor know complex query languages like SQL. Instead, users are only required to submit a list of keywords, and the system will return ranked answers based on their relevance to query keywords.

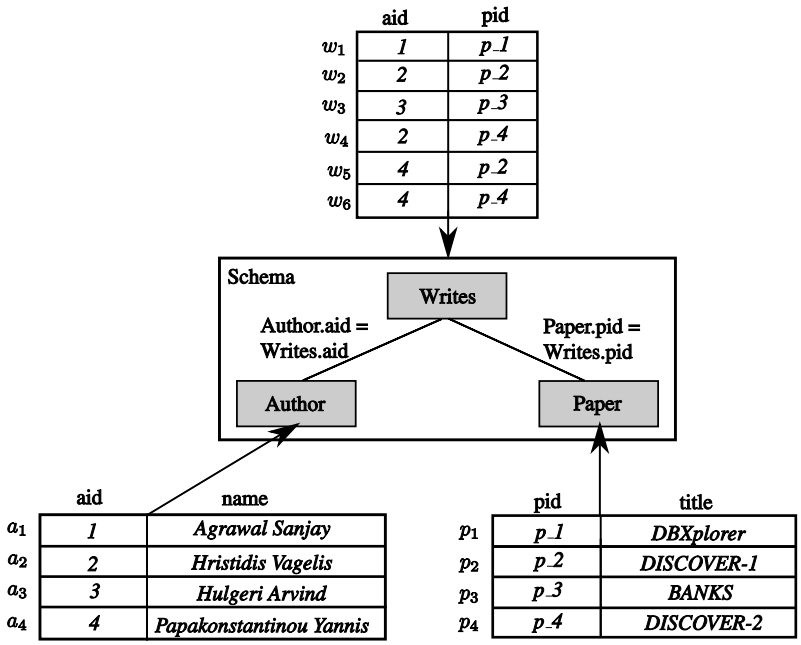


Figure 1.10 – The DBLP database

Consider the example of Figure 1.10, which illustrates a small subset of the DBLP database. If a user wants to get the papers co-authored by 'Hristidis Vagelis' and 'Papakonstantinou Yannis', she/he should learn the schema of the DBLP database first, and then she/he must write intricate SQL queries like this :

```
( SELECT   title
FROM       Paper, Writes, Author
WHERE      Paper.pid = Writes.pid
AND        Writes.aid = Author.aid
AND        Author.name = 'Hristidis Vagelis' )
∩
( SELECT   title
FROM       Paper, Writes, Author
WHERE      Paper.pid = Writes.pid
AND        Writes.aid = Author.aid
AND        Author.name = 'Papakonstantinou Yannis' )
```

Obviously, this model of search is too complicated for ordinary users. Several methods aim at decreasing this complexity by providing keyword search functionality over relational databases. With such functionality, a user can avoid writing an SQL query ; and she/he can just

submit a simple keyword query 'Hristidis Vagelis and Papakonstantinou Yannis' to the DBLP database. Examples include BANKS [BHN+02, KPC+05, HWYY07, KS06, DXW+07], DBXplore [ACD02] and DISCOVER [HP02, HGP03, LYMC06]. The former system [BHN+02, KPC+05, HWYY07, KS06, DXW+07] models the database as a graph and retrieves results by means of traversal, whereas the latter ones [ACD02, HP02, HGP03, LYMC06] exploit the database schema to compute the results.

BANKS

BANKS[20] [BHN+02] views the database as a directed weighted graph, where each node represents a tuple, and edges connect tuples that can be joined (e.g., according to primary-foreign key relationships). Node weight is inspired by prestige ranking such as PageRank [BP98] ; node that has large degree[21] get a higher prestige. Edge weight reflects the importance of the relationship between two tuples or nodes[22] ; lower edge weights correspond to greater proximity or stronger relationship between the involved tuples. At query time, BANKS employs a *backward* search strategy to search for results containing all keywords. A result is a tree of tuples (called tuple tree), that is, sets of tuples which are associated on their primary-foreign key relationships and contain all the keywords of the query. Figure 1.11 shows two tuple trees for query $q =$ 'Hristidis Vagelis and Papakonstantinou Yannis' on the example database of Figure 1.10. More precisely, BANKS constructs paths starting from each node (tuple) containing a query keyword (e.g., a_2 and a_4 in Figure 1.11) and executes a Dijkstra's single source shortest path algorithm for each one of them. The idea is to find a common vertex (e.g., p_2 and p_4 in Figure 1.11) from which a forward path exists to at least one tuple corresponding to a different keyword in the query. Such paths will define a rooted directed tree with the common vertex as the root containing the keyword nodes as leaves, which will be one possible answer (tuple tree) for a given query. The answer trees are then ranked and displayed to the user. The ranking strategy of BANKS is to combine nodes and edges weights in a tuple tree to compute a score for ranking.

Recently, [KPC+05, HWYY07] applied *bi-directional* (instead of *Backward*) search, which improves efficiency. [KS06, DXW+07] studied theoretical aspects of efficient query processing for top-k keyword queries in BANKS.

[20] Browsing ANd Keyword Searching

[21] A degree of a node is the number of edges incident to this node.

[22] The intuition is that a node that has many links with others has relative small possibility of having a close relationship to any of them, and thus edges incident on it have large weights.

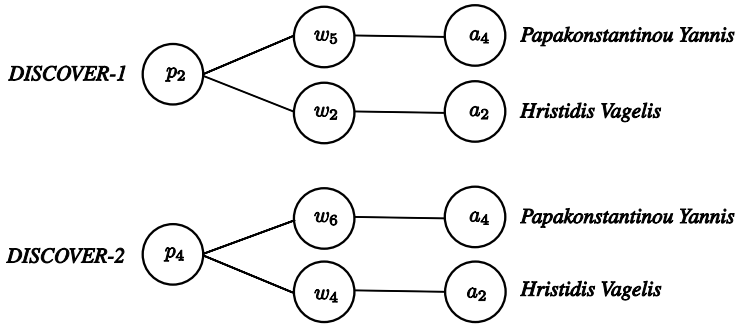


Figure 1.11 – Tuple trees for query $q =$ 'Hristidis Vagelis and Papakonstantinou Yannis'

DBXplorer

DBXplorer [ACD02] models the relational schema as a graph, in which nodes map to database relations and edges represent relationships, such as primary-foreign key dependencies. Given a query consisting of a set of keywords, DBXplorer first searches the symbol table to find the relations of the database that contain the query keywords. The symbol table serves as an inverted list and it is built by preprocessing the whole database contents before the search. Then, DBXplorer uses the schema graph to find join trees that interconnect these relations. A join tree is a subtree of schema graph that satisfies two conditions : one is that the relation corresponding to a leaf node contains at least one query keyword ; another is that every query keyword is contained by a relation corresponding to a leaf node. Thus, if all relations in a join tree are joined, the results might contain rows having all keywords. For each join tree a relevant SQL query is then created and executed. Finally, results are ranked and displayed to the user. The score function that DBXplorer uses to rank results is very simple. The score of a result is the number of joins involved. The rationale behind this simple relevance-ranking scheme is that the more joins are needed to create a row with the query keywords, the less clear it becomes whether the result might be meaningful or helpful.

DISCOVER

DISCOVER [HP02] also exploits the relational schema graph. It uses the concept of a candidate network to refer to the schema of a possible answer, which is a tree interconnecting the set of tuples that contain all the keywords, as in DBXplorer. The candidate network generation algorithm is also similar. However, DISCOVER can be regarded as an improvement of DBXplorer. In fact, it stores some temporary data to avoid re-executing joins that are common among candidate networks. DISCOVER, like DBXplorer, ranks results based on the number of joins

of the corresponding candidate network.

In [HGP03], the approach of DISCOVER is extended with information-retrieval techniques for ranking (e.g., $\mathtt{tf} * \mathtt{idf}$ [Sal89]). [HGP03] also proposed some efficient query-processing algorithms to obtain top-k results. Recently, the ranking method proposed in [HGP03] has been improved by Liu et al. [LYMC06] by using several refined weighting schemes.

Note that all afore-mentioned approaches (BANKS, DBXplore and DISCOVER) are useful to users who do not know SQL or are unfamiliar with the database schema. However, they present a semantic challenge because the metadata of attributes and relations that are in an SQL statement are lacking in keyword search. Furthermore, because a keyword may appear in any attributes and in any relations, the result set may be large and include many answers users do not need.

1.4 Discussion

Database search processes basically involve two steps, namely query formulation and query evaluation (Figure 1.12). In the query formulation step, the user formulates her/his information need (or *retrieval goal*) in terms of an SQL query. The query evaluation step runs this query against the database and returns data that match it exactly.

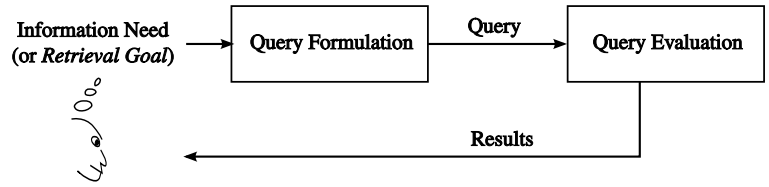


Figure 1.12 – Basic database searching process

Thus, formulating a query that accurately captures the user's retrieval goal is crucial for obtaining satisfactory results (i.e., results that are both useful and of manageable size for human analysis) from a database. However, it is challenging to achieve for several reasons :

- users may have *ill-defined retrieval goals*, i.e. , they do not really know what might be useful for them, merely an expectation that interesting data may be encountered if they use the database system (e.g., "I can't say what I want, but I will recognize it when I see it") ;
- even if users have *well-defined retrieval goals*, they may not know how to turn them into regular SQL queries (e.g., "I know what I want, but I don't know how to get it"). This

may be either because they are not familiar with the SQL language or with the database schema. It may also be due to an expressiveness limit of SQL. In fact, an information need is a mental image of a user regarding the information she/he wants to retrieve and it is difficult to capture it using an unnatural exact language such as SQL.

Note that even if users have *well-defined retrieval goals* and know how to formulate them in terms of SQL queries (e.g., "I know what I want and I know how to get it"), they may not obtain what they need (e.g., "I am not satisfied"). This occurs if the database they wish to access contains no data that satisfy their retrieval goals.

One can clearly notice that the *Many-Answers* and the *Empty-Answer* problems are immediate consequences of the above problems. In fact, if the user has an *ill-defined retrieval goal*, her/his queries are often very broad, resulting in too many answers ; otherwise, they are very specific and often return no answers.

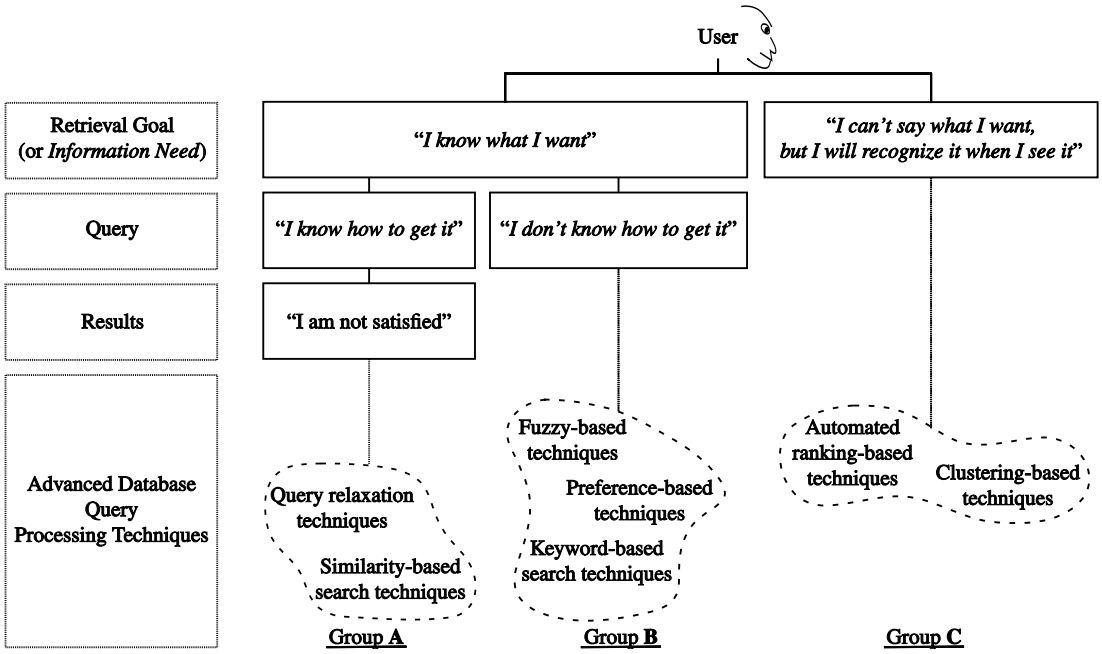


Figure 1.13 – A classification of advanced database query processing techniques

In Figure 1.13, we propose a classification of the techniques presented and discussed in this chapter. This classification is based on the situation or problem, from those mentioned above, that they are supposed to address.

The first category (Group **A** in Figure 1.13) contains query relaxation (Section 1.1.1) and similarity-based search (Section 1.1.2) techniques. These techniques address situations in which a user approaches the database with a query that exactly captures her/his retrieval goal but she/he

is delivered an empty result set. These techniques allow the database system to retrieve results that closely (though not completely) match the user's retrieval goal.

The second (Group **B** in Figure 1.13) contains preference-based (Section 1.3.1), fuzzy-based (Section 1.3.2) and keyword-based search (Section 1.3.3) techniques. These techniques provide human-oriented interfaces which allow users to formulate their retrieval goals in a more natural or intuitive manner. Note, however, that these techniques, while useful, do not help users to clarify or refine their retrieval goals ; therefore they are not designed for the problem of *ill-defined retrieval goal*.

Finally, the third category (Group **C** in Figure 1.13) contains automated ranking and clustering-based techniques. These techniques address situations in which a database user has an *ill-defined retrieval goal*. Automated ranking-based techniques (Section 1.2.1) first seek to clarify or approximate the retrieval goal. For this purpose, they use either past behavior of the user (derived from available workloads) or relevance feedback from the user. Then, they compute a score, by means of a similarity measure, of each answer that represents the extent to which it is relevant to the approximated user's retrieval goal. Finally, the user is provided with a ranked list, in descending order of relevance, of either all query results or only a top-k subset. The effectiveness of these approaches highly depends on their ability to accurately capture the user's retrieval goal, which is a tedious and time consuming task. Note that such approaches also bring the disadvantage of match homogeneity, i.e., the user is often required to go through a large number of similar results before finding the next different result. In contrast, clustering-based techniques (Section 1.2.2) assist the user to clarify or refine the retrieval goal instead of trying to learn it. They consist in dividing the query result set into homogeneous groups, allowing the user to select and explore groups that are of interest to her/him. However, such techniques seek to only maximize some statistical property of the resulting clusters (such as the size and compactness of each cluster and the separation of clusters relative to each other), and therefore there is no guarantee that the resulting clusters will match the meaningful groups that a user may expect. Furthermore, these approaches are performed on query results and consequently occur at query time. Thus, the overhead time cost is an open critical issue for such *a posteriori* tasks.

In the next chapter of this thesis, we focus on the *Many-Answers* problem that is critical for very large database and decision support systems. Thus, we investigate a simple but useful strategy to handle this problem.

CHAPTER 2

Knowledge-based Clustering of Result Set

Introduction

Database systems are being increasingly used for interactive and exploratory data retrieval. In such retrieval, user's queries often result in too many answers. Not all the retrieved items are relevant to the user ; typically, only a tiny fraction of the result set is relevant to her/him. Unfortunately, she/he often needs to examine all or most of the retrieved items to find the interesting ones. As discussed in Chapter 1, this phenomenon (commonly referred to as 'information overload') often happens when the user submits a 'broad' query, i.e., she/he has an *ill-defined retrieval goal*.

For example, consider a realtor database `HouseDB` with information on houses for sale in Paris, including their `Price`, `Size`, `#Bedrooms`, `Age`, `Location`, etc. A user who approaches that database with a broad query such as '`Price` $\in [150\text{k€}, 300\text{k€}]$' may be overloaded with a huge list of results, since there are many houses within this price range in Paris.

A well-established theory in cognitive psychology [Mil62, Man67] contends that humans organize items into logical groups as a way of dealing with large amounts of information. For instance, a child classifies his toys according to his favorite colors ; a direct marketer classifies his target according to a variety of geographic, demographic, and behavioral attributes ; and a real estate agent classifies his houses according to the location, the price, the size, etc. Furthermore, the *Cluster Hypothesis* [JR71] states that "closely associated items tend to be relevant to the same request". Therefore, clustering analysis [Ber06] which refers to partitioning data into dissimilar groups (or clusters) of similar items is an effective technique to overcome the problem of information overload. However, applying traditional clustering methods directly to the results of a user's query presents two major problems :

1. the first is related to *relevance*. Most clustering algorithms seek to only maximize some statistical property of the clusters (e.g., the size and compactness of each cluster and the

separation of clusters relative to each other), and therefore there is no guarantee that the resulting clusters will match the meaningful groups that a user may expect ;

2. the second is related to *scalability*. Clustering analysis is a time-consuming process, and doing it on the fly (i.e., at query time) may compromise seriously the response time of the system.

Now suppose that the user poses her/his query to a real estate agent. The estate agent often provides that user with better results than the ones obtained using traditional database systems, as she/he has a large amount of knowledge in the field of house buying. Let us briefly discuss two important features of the estate agent which account for this ability fitting the user's information need :

1. the first is her/his ability to organize, abstract, store and index her/his knowledge for future use. In fact, besides organizing her/his knowledge into groups [Mil62, Man67], the estate agent stores (in her/his memory) these groups as a *knowledge representation* and such groups often become an automatic response of her/him. This interesting statement comes from the central tenet of semantic network theory [Qui68, CQ69], which argues that information is stored in human memory as a network of linked concepts (e.g., the concept 'house' is related by the word 'is' to the concept 'home'). Moreover, psychological experiments [Mil56, Sim74, HBMB05] show that humans can deal with a large amount of information, exceeding their memory limitations, when such information are supplemented with additional features such as a relationship to a larger group (or concept). Hence, cognitive theories assume that humans arrange their knowledge in a hierarchical structure that describes groups at varying levels of specificity. Furthermore, Ashcraft [Ash94] found that humans assign meaningful words from natural language to groups and retrieve information by those words (i.e., group representatives) rather than blindly traversing all information ;
2. the second feature is her/his ability to assist the user to refine and clarify her/his information need as well as to make a decision. In fact, the estate agent establishes a dialog with the user during which she/he asks pertinent questions. Then, for each user's response (i.e., a new information need), the estate agent uses her/his knowledge to provide the user with concise and comprehensive information. Such information is retrieved by matching user query words with group representatives [Ash94] stored in her/his memory.

The SAINTETIQ [RM02, SPRM05, VRUM04] summarization technique provides a domain knowledge-based solution for clustering and querying large databases. We believe this technique has the potential to overcome the two previously mentioned problems (i.e., relevance and

scalability), since it emulates the interaction a user might have with a real estate agent to some extent. In fact, SAINTETIQ [RM02, SPRM05] first transforms raw data into high-level representations (summaries) that fit the user's perception of the domain, by means of linguistic labels (e.g., *cheap*, *reasonable*, *expensive*, *very expensive*) defined over the data attribute domains and provided by a domain expert or even an end-user. Then it applies a hierarchical clustering algorithm on these summaries to provide multi-resolution summaries (i.e., summary hierarchy) that represent the database content at different abstraction levels. The summary hierarchy can be seen as an analogy for *knowledge representation* estate agent. Furthermore, the summary hierarchy can be directly queried [VRUM04] to quickly provide the user with concise, useful and structured answers as a starting point for an online analysis. Each answer item describes a subset of the queried data in a human-readable form using linguistic labels. Moreover, answers of a given query are nodes of the summary hierarchy and every subtree rooted by an answer offers a 'guided tour' of a data subset to the user. Hence, this framework is intended to help the user iteratively refine her/his information need in the same way as done by the estate agent.

There are, however, two limitations that may restrict the effectiveness of this approach and which will be addressed in this chapter. First, since the summary hierarchy is independent of the query, the set of starting point answers could be large and, consequently, dissimilarity between items is susceptible to skew. It occurs when the pre-computed summary hierarchy is not perfectly adapted to the user query. To tackle this problem, we first propose a straightforward approach using the clustering algorithm of SAINTETIQ to optimize the high-level answers. The optimization requires post-processing and therefore incurs overhead time cost. Thus, we finally develop an efficient and effective algorithm that organizes answers based on the hierarchical structure of the pre-computed summary hierarchy, such that no post-processing task (but the query evaluation itself) have to be performed at query time. Furthermore, the current querying process cannot handle free vocabulary, i.e., summaries are exclusively queried by the linguistic labels used to build them. We propose a query rewriting process that deals with user-specific linguistic labels to overcome this limitation.

The rest of the chapter is organized as follows. First, we present the SAINTETIQ model and its properties and we illustrate the process with a toy example. Then, in Section 2.2 we detail the use of SAINTETIQ outputs in a query processing and we describe the formulation of queries and the retrieval of clusters. Thereafter, we discuss in Section 2.3 how such results help facing the *many-answers* problem. The algorithm that addresses the problem of dissimilarity (discrimination) between the starting point answers by rearranging them is presented in Section 2.4. Section 2.5 introduces an extension of the above process that allows every user to use

her/his own vocabulary when querying the database. An experimental study using real data is presented in Section 2.6. Section 2.7 concludes.

2.1 Overview of the SAINTETIQ System

In this section, we first introduce the main ideas of SAINTETIQ [RM02, SPRM05]. Then, we give useful definitions and properties regarding our proposal. Finally, we briefly discuss some other data clustering techniques, and argue that SAINTETIQ is more suitable for interactive and exploratory data retrieval.

2.1.1 A Two-Step Process

SAINTETIQ takes tabular data as input and produces multi-resolution summaries of records through an online mapping process and a summarization process.

2.1.1.1 Mapping Service

SAINTETIQ system relies on Zadeh's fuzzy set theory [Zad56], and more specifically on linguistic variables [Zad75] and fuzzy partitions [Rus69], to represent data in a concise form. The fuzzy set theory is used to translate records in accordance with a *Knowledge Base (KB)* provided by a domain expert or even an end-user. Basically, the operation replaces the original values of each record in the table by a set of linguistic labels defined in the *KB*. For instance, with a linguistic variable on the attribute `Income` (Figure 2.1), a value $t.Income = 95000€$ is mapped to *{0.3/tiny, 0.7/very small}* where 0.7 is a membership grade that tells how well the label *very small* describes the value 95000. Extending this mapping to all the attributes of a relation could be seen as mapping the records to a grid-based multidimensional space. The grid is provided by the *KB* and corresponds to the user's perception of the domain.

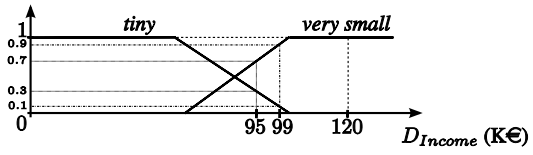


Figure 2.1 – Fuzzy linguistic partition defined on the attribute `Income`

Thus, tuples of Table 2.1 are mapped into three distinct grid-cells denoted by c_1, c_2 and c_3 in Table 2.2. *young* and *adult* are linguistic labels provided by the *KB* on the attribute `Age`

(Figure 2.2).

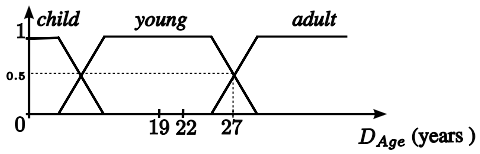


Figure 2.2 – Fuzzy linguistic partition defined on the attribute `Age`

Table 2.1 – Raw data (R)

ID	Age	Income
t_1	22	95000
t_2	19	99000
t_3	27	120000

Table 2.2 – Grid-cells mapping

Cell	Age	Income	Extent	tuple count
c_1	1.0/*young*	0.3/*tiny*	t_1, t_2	0.4=0.3+0.1
c_2	1.0/*young*	1.0/*very small*	t_1, t_2, t_3	2.1=0.7+0.9+0.5
c_3	0.5/*adult*	1.0/*very small*	t_3	0.5

In Table 2.2, 0.3/*tiny* says that *tiny* fits the data only with a small degree (0.3). The degree is computed as the maximum of membership grades of tuple values to *tiny* in c_1. Besides, the value 0.4 in the tuple count column gives the proportion of records in R that belong to c_1. It is computed as the sum of the membership grades of all the tuples within c_1. In fact, each tuple can belong to more than one cell with varying degrees of membership. The membership grade of the tuple t to the cell c is defined as the minimum of the membership grades of the t's attribute values to the corresponding linguistic labels in c. For instance, t_1 and t_2 are members of c_1 with membership grades of 0.3 and 0.1, respectively.

The *KB* leads to the point where tuples become indistinguishable and then are grouped into grid-cells such that there are finally many more records than cells. Each new (coarser) cell stores

a record count and attribute-dependant measures (min, max, mean, standard deviation, etc.). It is then called a *summary*.

2.1.1.2 Summarization Service

The summarization service (*SEQ*) is the second and the most sophisticated step of the SAINTETIQ system. It takes *grid-cells* as input and outputs a collection of summaries hierarchically arranged from the most generalized one (the root) to the most specialized ones (the leaves). Summaries are clusters of grid-cells, defining hyperrectangles in the multidimensional space. In the basic process, leaves are grid-cells themselves and the clustering task is performed on L cells rather than n tuples ($L << n$).

From the mapping step, cells are introduced continuously in the hierarchy with a top-down approach inspired of D.H. Fisher's Cobweb, a conceptual clustering algorithm [Fis87]. Then, they are *incorporated* into best fitting nodes descending the tree. Three more operators could be apply, depending on partition's score, that are *create*, *merge* and *split* nodes. They allow developing the tree and updating its current state. Figure 2.3 represents the summary hierarchy built from the cells c_1, c_2 and c_3 of Table 2.2. For the sake of simplicity, we have only reported the linguistic labels (intent) and the row IDs (extent) that point to tuples described by those linguistic labels.

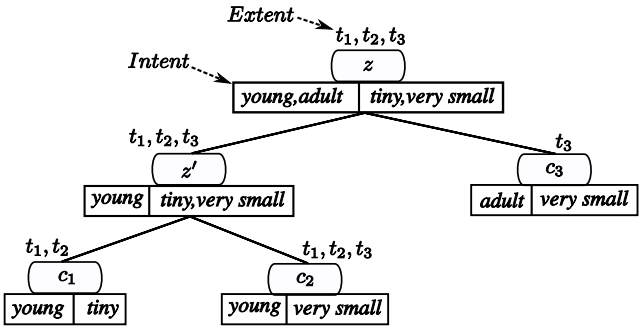


Figure 2.3 – Example of SAINTETIQ hierarchy

2.1.2 Features of the Summaries

In this section, we introduce the basic definitions and properties related to the SAINTETIQ model that are used throughout this thesis. For more details on SAINTETIQ, one can refer to [RM02, SPRM05].

Definition 2.1. ***Summary.***
Let $E = \{A_1, \ldots, A_N\}$ be a set of attributes and R a relation defined on the cartesian product of domains (D_{A_i}) of dimensions (A_i) in E. Assume that the N-dimensional space $A_1 \times A_2 \times \ldots \times A_N$ is equipped with a grid that defines basic N-dimensional areas, called cells, in E. The summary z of the relation R is the bounding box of the cluster of cells populated by records of R.

The above definition is constructive since it proposes to build generalized summaries (hyperrectangles) from cells that are specialized ones. In fact, it is equivalent to performing an *addition* on cells such that :

$$z = c_1 + c_2 + \ldots + c_m$$

where $\{c_1, \ldots, c_m\}$ is the set of m cells (summaries) covered by z. The operator $+$ stands for the union of the descriptions of cells to build the including hyperrectangle as a resulting summary.

A summary z is then an *intentional description* $\langle z.A_1, \ldots, z.A_N \rangle$ (or simply $\langle z.E \rangle$) associated with a set of tuples R_z as its *extent* and a set of cells L_z that are populated by records of R_z. Each $z.A_i$ is a description of the content of the summary z, expressed as a subset of the set of linguistic labels that partition the value domain of A_i. Hereafter, we shall use L_{R_z} and L_z interchangeably to denote the set of cells populated by records of R_z. These notations are illustrated in Figure 2.4 for the summary z (the root) of the hierarchy shown in Figure 2.3.

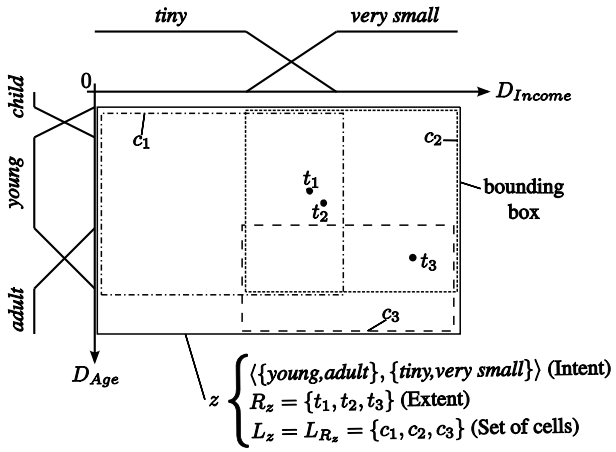


Figure 2.4 – Summary Notations

Thus, summaries are areas of E with hyperrectangle shapes provided by *KB*. They are nodes of the summary hierarchy built by the SAINTETIQ system.

Definition 2.2. ***Summary Hierarchy.***
A summary hierarchy H_R of R is a collection $\mathcal{Z}$ of summaries verifying :
- $\forall z, z' \in \mathcal{Z}, \quad z \preccurlyeq z' \Longleftrightarrow R_z \subseteq R_{z'}$;
- $\exists!\ z \in \mathcal{Z}$ *such that* $R_z = R$ *and* $L_z = \bigcup_{z' \in \mathcal{Z}} L_{z'}$ *(i.e., z is the summary of R).*

The relation over $\mathcal{Z}$ (i.e., $\preccurlyeq$) provides a generalization-specialization relationship between summaries. And assuming summaries are hyperrectangles in a multidimensional space, the partial ordering defines *nested summaries* from the larger one to single cells themselves.

In order to provide the end-user with a reduced set of representatives from the data, we need to extract a subset of the summaries in the summary hierarchy. The straightforward way of performing such a task is to define a summary partitioning.

Definition 2.3. ***Summary Partitioning.*** *The set* P *of leaves of every rooted sub-tree of the summary hierarchy H_R provides a partitioning of relation R.*

We denote by P_z the top-level partition of z in the summary hierarchy. It is then the most general partitioning of R_z we can provide from the hierarchy. Note that the most specialized one is the set of cells covering R_z, that is L_z.

A partitioning can be obtained *a posteriori* to meet user requirements regarding compression rate. For instance, general trends in the data could be identified in the very first levels of the tree, whereas precise information has to be looked for around leaf-level. Moreover, such partitioning verifies two basic properties : disjunction and coverage.

Property 2.1. ***Disjunction.*** *Summaries z and z' are disjoint iff* $\exists i \in [1..N],\ z.A_i \cap z'.A_i = \emptyset$.

Regarding this property, summaries of a partition do not overlap with each other, if we except *overlapping of adjacent fuzzy sets* (e.g., in Figure 2.4, if *tiny* and *very small* do not overlap at all, the cells c_1 and c_3 do not intersect).

Property 2.2. ***Coverage.*** $R = \bigcup_{z \in \mathrm{P}} R_z$

A partition P guarantees complete coverage of relation R since, by definition, representatives of every branch are included into P.

Moreover, SAINTETIQ process relies on an objective function, the *Summary Utility* (U) defined in [RM02], that guides local search through the space of possible summary trees. U is a combination of two well-known measures : *typicality* [RM75] and *contrast* [Tve77]. Those measures maximize between-summary dissimilarity and within-summary similarity. Thus, going

from the root (the highest partition) to the leaves (the lowest one) of the summary tree, the internal cohesion increases while the contrast between summaries decreases. Thanks to the aggregative function, trade-off values are found in the middle of the hierarchy and correspond to summaries with a high level of generalization while still informative. Figure 2.5 shows the evolution of those measures regarding the level of the hierarchy built on a CIC customer data set (see Section 2.6) that contains 33735 tuples. On this example, trade-off values are found at level 2. Note that this monotonicity relies on some cognitive assessments [RM75].

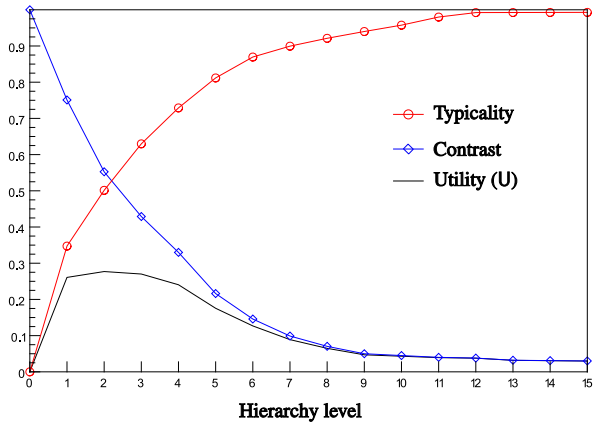


Figure 2.5 – Evolution of the Summary Utility U

2.1.3 Scalability Issues

In this section, we discuss the efficiency of the SAINTETIQ process and especially, its summarization service (*SEQ*). The mapping service will not be further discussed as it is a straightforward rewriting process.

The time complexity T_{SEQ} of the summarization service can be expressed as :

$$T_{SEQ}(L) = k_{SEQ} \cdot L \cdot \log_d L \in O(L \cdot \log L)$$

where L is the number of cells of the output hierarchy and d its average width. In the above formula, the coefficient k_{SEQ} corresponds to the set of operations performed to find the best learning operator (*create*, *merge* or *split*) to apply at each level of the hierarchy, whereas $\log_d L$ is an estimation of the average depth of this hierarchy.

Note that the number of leaves L is bounded by p^N (i.e., the size of the grid-based multi-dimensional space) where p represents the average number of linguistic labels defined for each feature in E. Of course, the exact number will greatly depend on the data set, and more specifically, on the existing correlations between attribute values. For example, in a bank's database

with attributes `Occupation` and `Income`, it is likely that we will not find the combination of *unemployed* and *enormous* income. Thus, in 'real life' situations, L is expected to be far smaller than p^N [SPRM05].

2.1.4 Discussion about SAINTETIQ

Cluster analysis is one of the most useful tasks in data mining [MR05] process for discovering groups and identifying interesting distributions and patterns in the underlying data. The clustering problem is about partitioning a given data set into groups (clusters) such that the data points in a cluster are more similar to each other than to points in different clusters.

Up to now, many clustering methods [Ber06] have been proposed and, among them, grid-based clustering methods (e.g., STING [WYM97], BANG [SE98], WaveCluster [SCZ00], etc.). Grid-based clustering methods first partition the data by applying a multidimensional grid structure on the feature space. Second, statistical information (e.g., min, max, mean, standard deviation, distribution) is collected for all the database records located in each individual grid cell and clustering is performed on populated cells to form clusters. These methods have been proved as valuable tools for analyzing the structural information of very large databases. One of the most appealing factors is the excellent runtime behavior. In fact, their processing time only depends on the number of populated cells L which is usually much less than the number of database records n ($L << n$) [Ber06].

The SAINTETIQ model, besides being a grid-based clustering method, has many other advantages that are relevant for achieving the targeted objective of this thesis. First, SAINTETIQ uses prior domain knowledge (the *Knowledge Base*) to guide the clustering process, and to provide clusters that fit the user's perception of the domain. As mentioned in the introduction of this chapter, this distinctive feature differentiates it from other grid-based clustering techniques which attempt to only maximize some statistical property of the clusters, and therefore there is no guarantee that the resulting clusters will match the meaningful groups that a user may expect. Second, the flexibility in the vocabulary definition of *KB* leads to clustering schemas that have two useful properties : (1) the clusters have 'soft' boundaries, in the sense that each record belongs to each cluster to some degree, and thus undesirable threshold effects that are usually produced by crisp (non-fuzzy) boundaries are avoided ; (2) the clusters are presented in a user-friendly language (i.e., linguistic labels) and hence the user can determine at a glance whether a cluster's content is of interest. Finally, SAINTETIQ applies a conceptual clustering algorithm for partitioning the incoming data in an incremental and dynamic way. Thus, changes in the database are reflected through such an incremental maintenance of the complete hierar-

chy [SPRM05].

Of course, for new application, the end-user or the expert has to be consulted to create linguistic labels as well as the fuzzy membership functions. However, it is worth noticing that, once such knowledge base is defined, the system does not require any more setting. Furthermore, the issue of estimating fuzzy membership functions has been intensively studied in the fuzzy set literature [Gal08], and various methods based on data distribution and statistics exist to assist the user designing trapezoidal fuzzy membership functions.

As one can observe, a SAINTETIQ summary hierarchy is similar to a multidimensional index structure. Indexes based on tree structures are widely used for organizing multidimensional large data sets and accelerating access to them. The most popular are R-Tree [Gut88] and its variants R*-tree [BKSS90], X-Tree [BKK96], SS-Tree [WJ96], etc. R-Tree-based approaches partition the data domain into buckets (i.e., intervals in several dimensions) and build a height-balanced tree that represents the data as nested structured buckets of increasing resolution. Each leaf node is a bucket containing at most M pointers to the tuples it covers and each inner node contains at most M pointer to lower nodes in the tree structure. M is an input parameter which depends on the hard disk drive's properties (e.g., capacity and sector size). Theses approaches, even though they are very efficient for multidimensional data querying, are not very useful in the context of data discovery, analysis and exploration. Indeed, the buckets (grid-cells) depends on the distribution of the record values rather than the user's perception of the domain. Furthermore, the grouping process depends on the parameter M and consequently, tuples within each group do not meet the cluster hypothesis "closely associated tuples tend to be relevant to the same query".

In the next section, we present the querying mechanism [VRUM04] that allows users to efficiently access the hierarchical summaries produced by SAINTETIQ.

2.2 Querying the SAINTETIQ Summaries

In an exploratory analysis of a massive data set, users usually have only a vague idea of what they could find in the data. They are then unable to formulate precise criteria to locate the desired information. The querying mechanism [VRUM04] presented here allows such users to access a database (previously summarized) using vague requirements (e.g., *cheap*) instead of crisp ones (e.g., $[100\text{k}€, 200\text{k}€]$). In fact, users only need to select the right criteria from an existing set of linguistic labels defined on each attribute domain in the *KB*, to filter a set of

clusters (summaries) that can then be browsed to find potentially interesting pieces of information. However, choosing the linguistic labels from a controlled vocabulary compels the user to adopt a predefined categorization materialized by the grid-cells. In section 2.5, we deal with user-specific linguistic labels to overcome this pitfall.

In this section, we first introduce a toy example that will be used throughout that chapter. Then, we present all aspects of the querying mechanism from the expression and meaning of a query to its matching against summaries.

2.2.1 Running Example

To illustrate the querying mechanism, we introduce here a sample data set R with 30 records ($t_1 - t_{30}$) represented on three attributes : `Price`, `Size` and `Location`. We suppose that {*cheap* (ch.), *reasonable* (re.), *expensive* (ex.), *very expensive* (vex.)}, {*small* (sm.), *medium* (me.), *large* (la.)} and {*downtown* (dw.), *suburb* (su.)} are sets of linguistic labels defined respectively on attributes `Price`, `Size` and `Location`. Figure 2.6 shows the summary hierarchy H_R provided by SAINTETIQ performed on R.

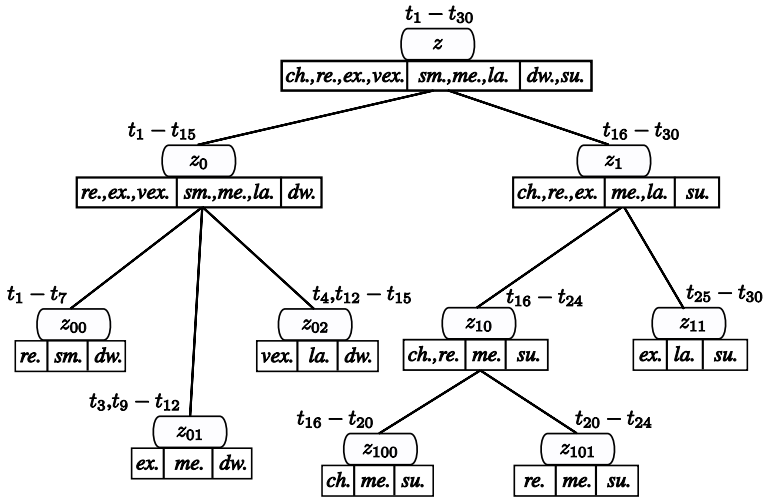


Figure 2.6 – Summary hierarchy H_R of the data set R

2.2.2 Expression of Queries

The querying mechanism described here intends to evaluate questions such as "find *medium* or *large* houses in the *suburb*". In the prototype developed for querying, questions are expressed

using a user-friendly interface that composes the corresponding query in an SQL-like language. For the previous question, the query formed is :

Example 2.1.

$$\texttt{Q1} \equiv \begin{array}{ll} \text{SELECT} & * \\ \text{FROM} & R \\ \text{WHERE} & \texttt{Location IN } \{\textit{suburb}\} \\ \text{AND} & \texttt{Size IN } \{\textit{medium} \text{ OR } \textit{large}\} \end{array}$$

In the following, we denote by X the set of attributes specified in the user's input query Q and by Y the set of missing attributes which is the complement of X relatively to R : $X \cup Y = R$ and $X \cap Y = \emptyset$. Further, for each attribute $A \in X$, Q_A denotes the set of required features defined on attribute A.

Example 2.2. *Consider the query* `Q1` *of Example 2.1. We have :*

X =*{*`Location`, `Size`*}*
Y =*{*`Price`*}*
$Q_{Location}$ =*{suburb}*
Q_{Size} =*{medium,large}*

When users formulate a query, they expect data satisfying the selection criteria to be put forward. The interpretation we adopted is that user's queries are under a conjunctive form : modalities on an attribute are connected with an OR and attributes are connected with an AND operator. Hence, the query `Q1` is interpreted as "find records in R that are described by *suburb* on the attribute `Location` AND by either *medium* OR *large* on the attribute `Size`". The AND operator is used because the process should select only data that comply with the characterization on both `Location` and `Size`. On the other hand, the use of the OR operator is due to a data constraint : database records are single-valued tuples.

2.2.3 Evaluation of Queries

This section deals with matching one particular summary against a query to decide whether it corresponds to that query and can then be considered as a result. The query is transformed into a logical proposition P used to qualify the link between the summary and the query. P is under a conjunctive form in which all linguistic labels appear as literals. In consequence, each set of linguistic labels yields one corresponding clause (see Example 2.3).

Example 2.3. *The logical proposition corresponding to the query* `Q1` *is :*

$$P_1 = (\textit{medium} \vee \textit{large}) \wedge \textit{suburb}.$$

Let v be a valuation function. It is obvious that the valuation of P depends on the summary z : a literal d in P is positively valuated ($v(d) = \texttt{TRUE}$) if and only if d appears in z. So we denote by $v(P(z))$ the valuation of proposition P in the context of z. An interpretation of P relatively to query `Q` leads to discarding summaries that do not satisfy P. But, as shown in Example 2.4, some summaries might satisfy P and yet not match the intended semantics of the query.

Example 2.4. *Table 2.3 shows the characteristics of houses for sale covered by a summary z' along with z' itself. If z' is tested for conformance with* `Q1` *(Example 2.3), we can see that $v(P_1(z')) =$* `TRUE`*, but nowhere can one find a house responding to query* `Q1`*.*

Table 2.3 – Example of linguistic label combination

ID_Sum	Size	Location
c_1	*small*	*suburb*
c_2	*medium*	*downtown*
c_3	*large*	*downtown*
z'	{*small*,*medium*,*large*}	{*downtown*,*suburb*}

Recall that $z.A$ denotes the set of linguistic labels that appear in a summary z on attribute A. When matching z with a query Q, three cases might occur :

a : no correspondence. $v(P(z)) = \texttt{FALSE}$. For one attribute or more, z has no required feature, i.e., it shows none of the linguistic labels mentioned in query Q : $\exists A \in X,\ z.A \cap Q_A = \emptyset$;

b : exact correspondence. The summary z matches the query Q semantics. It is considered as a result. The following expression holds : $\forall A \in X,\ z.A \subseteq Q_A$;

c : no decision can be made. There is one attribute A for which z exhibits one or many linguistic labels besides those strictly required (i.e., those in Q_A) : $\exists A \in X,\ z.A \setminus Q_A \neq \emptyset$.

The presence of required features in each attribute of z suggests, but does not guarantee, that results may be found in the subtree rooted by z. Exploration of the subtree is necessary to retrieve possible results : for each branch, it will end up in situations categorized

by case **a** or case **b**. Thus, at worst at leaf level, an exploration leads to accepting or rejecting summaries ; the indecision is always solved.

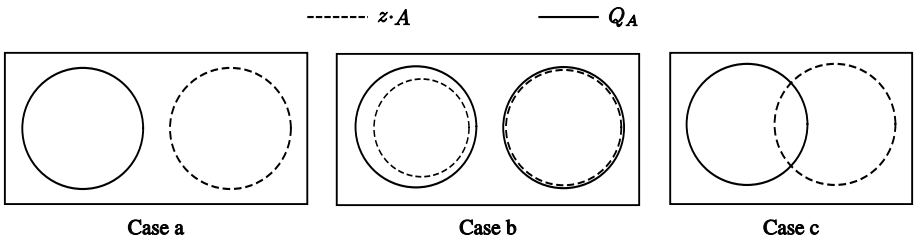


Figure 2.7 – Comparison of linguistic label sets $z.A$ and Q_A (cf. [VRUM04])

The situations stated above reflect a global view of the matching of a summary z with a query Q. They can also be interpreted, from a crisp set point of view, as a combination of comparisons, still involving $z.A$ and Q_A, concerning one required attribute A. Figure 2.7 shows all comparisons using a set representation with $z.A$ symbolized by a dashed circle and Q_A by a solid circle.

2.2.4 Search Algorithm

The Explore-Select Algorithm (*ESA*) applies the matching procedure from the previous section over the whole set of summaries, organized in a hierarchy, to select relevant summaries. Since the selection should take into account all summaries that correspond to the query, the exploration of the hierarchy is complete. Algorithm 1 (shown on the next page) describes *ESA* with the following assumptions :

- the function returns a list of summaries ;
- function Corr symbolizes the matching test reported in Section 2.2.3 ;
- operator '+' performs a list concatenation of its arguments ;
- function Add is the classical constructor for lists, it adds an element to a list of the suitable type ;
- L_{res} is a local variable.

The *ESA* algorithm is based on a depth-first search and relies on a property of the hierarchy : the generalization step in the SAINTETIQ model guarantees that any label that exists in a node of the tree also exists in each parent node. Inversely, a label is absent from a summary's intent if and only if it is absent from all subnodes of this summary. This property of the hierarchy permits branch cutting as soon as it is known that no result will be found. Depending on the query, only a part of the hierarchy is explored.

Algorithm 1 Function Explore-Select(z, Q)

$L_{res} \leftarrow \langle\rangle$
if Corr(z, Q) = *indecisive* **then**
  for all child node z_{child} of z **do**
    $L_{res} \leftarrow L_{res}+$ Explore-Select(z_{child}, Q)
  end for
else
  if Corr(z, Q) = *exact* **then**
    Add(z, L_{res})
  end if
end if
return L_{res}

Thus, results of `Q1` (Example 2.1), when querying the hierarchy shown in Figure 2.6, look like :

Table 2.4 – `Q1` results

Id_Sum	Price	Size	Location
z_1	*cheap* *reasonable* *expensive*	*medium* *large*	*suburb*

In this case, the *ESA* algorithm first confronts z (root) with `Q1`. Since no decision can be made, `Q1` is respectively confronted to z_0 and z_1, the children of z. The subtree rooted by z_0 is then ignored because there is no correspondence between `Q1` and z_0. Finally z_1 is returned because it exactly matches `Q1`. Thus, the process tests only 30% of the whole hierarchy.

Note that, the set of records R_{z_1} summarized by z_1 can be returned if the user requests it (SHOWTUPLES option). This is done by simply transforming the intent of z_1 into a query q_{z_1} and sending it as a usual query to the database system. The WHERE clause of q_{z_1} is generated by transforming the linguistic labels (fuzzy sets) contained in the intent of z_1 into crisp ones. In other words, each linguistic label l on an attribute A is replaced by its support, i.e., the set of all values in the domain of A (D_A) that belong to l with non-zero membership : *support*$(l) = \{u \in D_A \mid \mu_l(u) > 0\}$. Then, the obtained crisp criteria on each summary's attribute are connected with OR operator and summary's attributes are connected with AND

operator to generate the WHERE clause. Thus, performing a SHOWTUPLES operation takes advantage of the optimization mechanisms that exist in the database system. Furthermore, tuples covered by z_1 can also be sorted, thanks to their satisfaction degrees to the user's query, using an overall satisfaction degree. We assign to each tuple the degree to which it satisfies the fuzzy criteria of the query. Usually, the minimum and maximum functions stand for the conjunctive and disjunctive connectives. There are many propositions in the literature for defining aggregation connectives [KMP00].

The *ESA* algorithm is particularly efficient. In the worst case (exploration of the hierarchy is complete), its time complexity is given by :

$$T_{ESA}(L) = \varepsilon \cdot \frac{L-1}{d-1} \in O(L)$$

where L is the number of leaves (cells) of the queried summary hierarchy, d its average width and coefficient ε corresponds to the time required for matching one summary in the hierarchy against the query. In the above formula, $[\frac{L-1}{d-1}]$ gives an estimation of the number of nodes in the summary hierarchy.

In the following section, we discuss how *ESA*'s answers help facing the *many-answers* problem.

2.3 Multi-Scale Summarized Answers

Given a query Q, the *ESA* algorithm produces a set of clusters (summaries) from a SAINTETIQ hierarchy instead of a list of tuples $tset(Q)$. Each answer item z describes a subset of the query result set $tset(Q)$. Hence, the user can unambiguously determine whether R_z contains relevant tuples only by looking at the intentional description of z and particularly on unspecified attributes (Y) because all answer tuples satisfy the specified conditions related to attributes in X. Three cases might occur :

1 : summary z doesn't fit the user's need. It means that for at least one unspecified attribute $A \in Y$, all linguistic labels (the set $z.A$) are irrelevant to the user. For instance, consider the `Q1` results (Table 2.4). z_1 doesn't contain any relevant tuple if the user is actually looking for *very cheap* houses because *very cheap* $\notin z_1.Price$. In other words, none of the records in R_{z_1} is mapped to *very cheap* on attribute `Price` and consequently a new broad query with less selective conditions may be submitted or the task may be abandoned (we denote this with the IGNORE option).

2 : summary z exactly fits the user's need. It means that for each $A \in Y$, all linguistic labels in $z.A$ are relevant to the user. Assume that the user is interested in *cheap*, *reasonable* as well as *expensive* houses. Thus, all tuples contained in R_{z_1} are relevant to her/him. In such cases, she/he uses SHOWTUPLES option to access tuples stored in R_{z_1}.

3 : summary z partially fits the user's need. In this case, there is at least one attribute $A \in Y$ for which $z.A$ exhibits too many linguistic labels w.r.t. the user's requirement. For instance, the set R_{z_1} partially matches the needs of a user who is looking for *cheap* as well as *reasonable* houses because R_{z_1} contains also tuples that are mapped to *expensive* on attribute `Price`. In this case, a new query with more selective conditions (e.g., `Price` IN {*cheap* OR *reasonable*}) may be submitted or a new clustering schema of the set R_z, i.e., which allows to examine more precisely the dataset, is required. Since z is a subtree of the summary hierarchy, we present to the user the children of z (SHOWCAT option). Each child of z represents only a portion of tuples in R_z and gives a more precise representation of the tuples it contains. For example, $\{z_{10}, z_{11}\}$ is a partitioning of R_{z_1} into two subsets $R_{z_{10}}$ and $R_{z_{11}}$; z_{10} exactly fits user needs. Since the entire tree is pre-computed, no clustering at all would have to be performed at feedback time.

More generally, a set of summaries or clusters $S = \{z_1 \dots z_m\}$ is presented to the user as a clustering schema of the query result $tset(Q)$. The three options IGNORE (case 1), SHOWTUPLES (case 2) and SHOWCAT (case 3) give the user the ability to browse through the S structure (generally a set of rooted subtrees), exploring different datasets in the query results and looking for potentially interesting pieces of information. Indeed, the user may navigate through S using the basic exploration model given below :

i) start the exploration by examining the intensional description of $z_i \in S$ (initially $i = 0$) ;
ii) if case 1, ignore z_i and examine the next cluster in S, i.e., z_{i+1} ;
iii) if case 2, navigate through tuples of R_{z_i} to extract every relevant tuple and thereafter, go ahead and examine z_{i+1} ;
iiii) if case 3, navigate through children of z_i, i.e., repeat from step (i) with S, the set of children of z_i. More precisely, examine the intensional description of each child of z_i starting from the first one and recursively decide to ignore it or examine it (SHOWTUPLES to extract relevant tuples or SHOWCAT option for further expansion). At the end of the exploration of the children of z_i, go ahead and examine z_{i+1}.

For instance, suppose a user is looking for *medium*, *large* as well as *expensive* houses in the *suburb* but issues the broad query `Q1` (Example 2.1) : "find *medium* or *large* houses in the

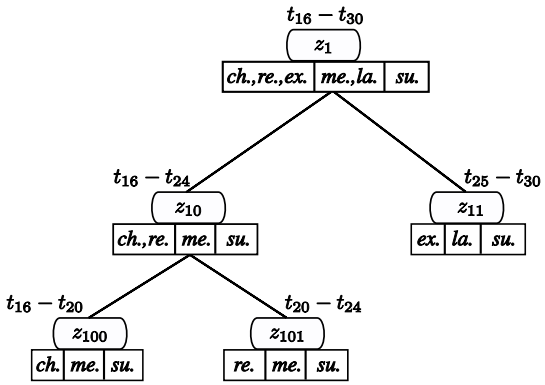


Figure 2.8 – Summary z_1

suburb". The set of summaries S presented to that user is $\{z_1\}$, where z_1 is a subtree (Figure 2.8) in the pre-computed summary hierarchy shown in Figure 2.6. In this situation, the user can explore the subtree rooted by z_1 as follows to reach relevant tuples : analyze the intent of z_1 and explore it using SHOWCAT option, analyze the intent of z_{10} and ignore it, analyze the intent of z_{11} and use SHOWTUPLES option to navigate through the tuples in $R_{z_{11}}$ (i.e., $t_{25} - t_{30}$) to identify each relevant tuple.

Note that when the set $S = \{z\}$ is a singleton, i.e., z is a node of the pre-computed clustering tree, its exploration is straightforward. Indeed, given a summary of the tree rooted by z that the user wishes to examine more closely (SHOWCAT option), its children are well separated since SAINTETIQ is designed to discover summaries (clusters) that locally optimize the objective function U. Furthermore, the number of clusters presented to the user, at each time, is small ; the highest value is equal to the maximum width of the pre-computed tree.

However, since the summary hierarchy is independent of the query, the set of starting point answers S could be large and consequently dissimilarity between summaries is susceptible to skew. It occurs when the summary hierarchy is not perfectly adapted to the user query. In this situation, it is hard for the user to separate the interesting summaries from the uninteresting ones, thereby leading to potential decision paralysis and a wastage of time and effort.

In the next section, we propose an original rearranging query results algorithm to tackle this problem.

2.4 Rearranging the Result Set

The problem of discrimination (dissimilarity) between *ESA*'s results occurs when these results are scattered over the queried summary hierarchy. This situation is illustrated in Figure 2.9, where the set of summaries $S = \{z_{00}, z_{01}, z_{1000}, z_{101}, z_{11}\}$ is returned by *ESA* as the result of a query Q over the summary hierarchy H.

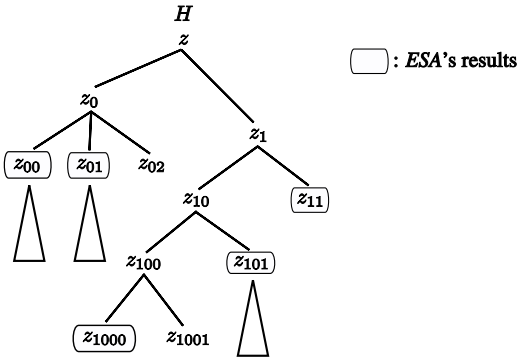


Figure 2.9 – An example of *ESA*'s results

A straightforward way to address this problem would be to, first, execute the SAINTETIQ summarization service (*SEQ*) on the cells populated by records of $tset(Q)$, i.e., the cells covered by summaries of S. Then, we present to the user the top-level partition of the result tree. We will refer to this approach as *ESA+SEQ* (i.e., search step followed by summarization step) for the remainder of this chapter.

Size reduction and discrimination between items in S are clearly achieved at the expense of an overhead computational cost. Indeed, the search step time complexity T_{ESA} is in $O(L)$ where L is the number of leaves of the queried summary hierarchy (see Section 2.2.4). Furthermore, the summarization step time complexity T_{SEQ} is in $O(L' \cdot \log L')$ with L' the number of cells populated by answer records (see Section 2.1.3). Therefore, the global time complexity $T_{ESA+SEQ}$ of the *ESA+SEQ* approach is in $O(L \cdot \log L) : L \simeq L'$ since the query is expected to be broad. Thus, *ESA+SEQ* doesn't fit the querying process requirement (see experimental results in Section 2.6), that is to quickly provide the user with concise and structured answers.

To tackle this problem, we propose an algorithm coined Explore-Select-Rearrange Algorithm (*ESRA*) that rearranges answers, based on the hierarchical structure of the queried summary hierarchy, before returning them to the user. The main idea of this approach is rather simple. It starts from the summary partition S (a clustering schema of the query results) and produces a sequence of clustering schemas with a decreasing number of clusters at each step.

Each clustering schema produced at each step results from the previous one by merging the 'closest' clusters into a single one. Similar clusters are identified thanks to the hierarchical structure of the pre-computed summary hierarchy. Intuitively, summaries which are closely related have a common ancestor lower in the hierarchy, whereas the common ancestor of unrelated summaries is near the root. This process stops when it reaches a single hyperrectangle (the root $z^\star$). Then, we present to the user the top-level partition (i.e., children of $z^\star$) in the obtained tree instead of S.

For instance, when this process is performed on the set of summaries $S = \{z_{00}, z_{01}, z_{1000}, z_{101}, z_{11}\}$ shown in Figure 2.9, the following sequence of clustering schemas is produced :

1	z_{00}	z_{01}	z_{1000}	z_{101}	z_{11}	$\leftarrow S$
2	$z_{00} + z_{01}$		$z_{1000} + z_{101}$		z_{11}	
3	$z_{00} + z_{01}$		$z_{1000} + z_{101} + z_{11}$			
4	$z_{00} + z_{01} + z_{1000} + z_{101} + z_{11}$					

The hierarchy H' obtained from the set of query results S is shown in Figure 2.10. Thus, the partition $\{z', z'''\}$ is presented to the user instead of S. This partition has a small size and defines well separated clusters. Indeed, all agglomerative methods, including the above rearranging process, have a monotonicity property [HTF01] : the dissimilarity between the merged clusters is monotonically increasing with the level. In the above example, it means that the dissimilarity value of the partition $\{z', z'''\}$ is greater than the dissimilarity value of the partition $\{z_{00}, z_{01}, z_{1000}, z_{101}, z_{11}\}$.

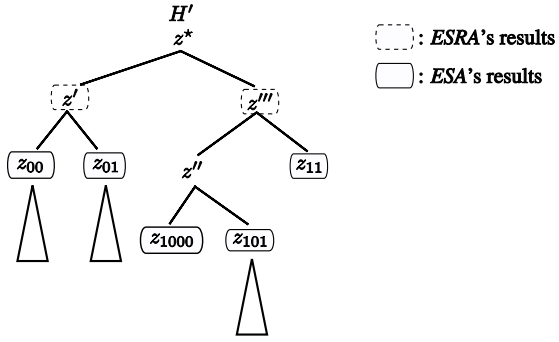


Figure 2.10 – Rearranging *ESA*'s results

Algorithm 2 (shown on the next page) describes *ESRA*. It is a modified version of *ESA* (Algorithm 1) with the following new assumptions :

- it returns a summary ($z^\star$) rather than a list of summaries ;
- function AddChild appends a node to caller's children ;
- function NumberofChildren returns the number of caller's children ;

- function uniqueChild returns the unique child of the caller ;
- function BuildIntent builds caller's intent (hyperrectangles) from intents of its children ;
- $Zres$ and Z' are local variables of type summary.

Algorithm 2 Function Explore-Select-Rearrange(z, Q)

```
Zres ← Null
Z' ← Null
if Corr(z, Q) = indecisive then
  for all child node z_child of z do
    Z' ← Explore − Select − Rearrange(z_child, Q)
    if Z' ≠ Null then
      Zres.addChild (Z')
    end if
  end for
  if Zres.NumberofChildren() > 1 then
    Zres.BuildIntent()
  else
    Zres ← Zres.uniqueChild()
  end if
else
  if Corr(z, Q) = exact then
    Zres ← z
  end if
end if
return Zres
```

The *ESRA* cost is only a small constant factor γ larger than that of *ESA*. In fact, the rearranging process is done at the same time the queried summary hierarchy is being scanned. It means that no post-processing task (but the query evaluation itself) have to be performed at query time. More precisely, the time complexity of the *ESRA* Algorithm is in the same order of magnitude (i.e., $O(L)$) than the *ESA* Algorithm :

$$C_{ESRA}(L) = (\gamma + \varepsilon) \cdot \frac{L-1}{d-1} \ \in \ O(L)$$

where the coefficient γ is the time cost for the additional operations (addChild, uniqueChild and BuildIntent). L is the number of leaves (cells) of the queried summary hierarchy and d its

average width.

A limitation of our proposal relies in the fact that *ESRA* restrains the user to queries using a controlled vocabulary (i.e., the summary vocabulary). Indeed, consider a user looking for *cheap* houses. What is meant by *cheap* can vary from one user to another one, or from one kind of users to another one (e.g., *cheap* does not have the same meaning for house buyers and for apartment tenants). In the following, we deal with user-specific linguistic labels to overcome this problem.

2.5 Extension to any Fuzzy Predicate

We aim at allowing the user to formulate conditions in selection queries in her/his own language. Regarding the query cost and accuracy, the ideal solution consists in building a summary hierarchy for every user such that queries and summaries share the same vocabulary. Thus, *ESRA* can be used in its current state. But maintaining *user-specific summaries* is hardly conceivable in a system with multiple users. Two alternatives are then envisaged.

The first one is to consider *group profiles* in order to reduce the number of managed summaries (e.g., one for buyers and the other for tenants). In this case, *ESRA* can also be used directly since users formulate queries within the vocabulary of their groups. In this approach, users would subscribe to groups that share the same (or similar) vocabulary as theirs. In addition, they have to be familiar with their group's linguistic labels before using them accurately. As a result, this option is not much more convenient than using ad-hoc linguistic labels predetermined by a domain expert. Moreover, it only transposes the problem of user-specific summaries maintenance to group-specific ones.

The second alternative, investigated in this section, consists in building only one SAINTETIQ summary hierarchy using an ad-hoc vocabulary, and querying it with user-specific linguistic labels. In fact, we consider one summary hierarchy of the database which is created using *a priori* defined linguistic labels that constitute the summary vocabulary. At the same time, the user defines her/his own linguistic labels to formulate free fuzzy predicates within selection queries and then searches the summary hierarchy thanks to a vocabulary mapping step.

2.5.1 Query Rewriting

Since the vocabulary of the query predicates is different from the one in the summaries, the query must be rewritten before it can be answered using *ESRA*. The query rewriting process

proposed here uses a fuzzy set-based mapping operation to translate query predicates from the user-specific vocabulary to the summary language.

Consider a fuzzy query expressed using user-specific linguistic labels. Each linguistic label l_u, defined on an attribute A with domain D_A, is rewritten by the smallest superset of linguistic labels l_s taken from the summary vocabulary, using the usual fuzzy intersection :

$$l_u \cap l_s \neq \emptyset \quad \text{iff} \quad \exists x \in D_A,\ \min(\mu_{l_u}(x), \mu_{l_s}(x)) > 0$$

For instance, in the query "`Location` IN {*well-located*}", the user-specific label *well-located* is rewritten with the two summary linguistic labels *downtown* and *town* (Figure 2.11). Thus, the resulting query is "`Location` IN {*downtown* OR *town*}".

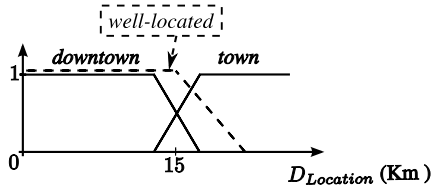


Figure 2.11 – The user label *well-located* (dashed line) is rewritten with the summary labels *downtown* and *town*

In this scenario, the user query Q has been rewritten in a *conservative* manner so that "nothing is lost". Thus, an empty set of summaries ($S = \emptyset$) as an answer guarantees that there is really no result tuples for the query, i.e., $tset(Q) = \emptyset$. A non-empty set $S = \{z_1 \dots z_m\}$ denotes the possibility that answer tuples exist, but nothing can be guaranteed, i.e., $tset(Q) \subseteq \bigcup_{1 \leq i \leq m} R_{z_i}$. Of course, tuples of $\bigcup_{1 \leq i \leq m} R_{z_i} \setminus tset(Q)$ do not satisfy Q. However, they are close to Q and consequently interesting since the user is in an exploratory analysis and tries to discover the content of the database and the choices available to her/him.

In some cases, it could be more suitable to guarantee that answers exist instead of guaranteeing empty answers only. To achieve this, the user linguistic label l_u is rewritten by the subset of linguistic labels l_s taken from the summary vocabulary, using the usual fuzzy inclusion :

$$l_s \subseteq l_u \quad \text{iff} \quad \forall x \in D_A,\ \mu_{l_s}(x) \leq \mu_{l_u}(x)$$

For instance, the linguistic label *well-located* (Figure 2.11) is only mapped to *downtown*. Thus, in this case, the resulting query is "`Location` IN {*downtown*}".

In this second rewriting scenario, a non-empty set $S = \{z_1 \dots z_m\}$ as an answer guarantees that all tuples covered by its summaries are really in the result of the query, i.e., $\bigcup_{1 \leq i \leq m} R_{z_i} \subseteq$

$tset(Q)$. However, there could be others that satisfy predicates and do not belong to S, i.e., $tset(Q) \setminus \bigcup_{1 \leq i \leq m} R_{z_i}$.

Note that both scenarios can be used together for the same query, as one guarantees null answers and the other guarantees non-null answers. This dual approach can be viewed as reasoning with both Possibility and Necessity [DPU02] of the query result. Moreover, these two scenarios are the boundaries of a spectrum of possible strategies : the generalized expression of the rewriting process consists in defining a similarity measure σ and a threshold τ to decide whether or not the mapping of two linguistic labels is valid. Thus, a user label l_u is mapped to a summary label l_s if and only if $\sigma(l_u, l_s)$ is greater than or equal to τ.

In this work, we use the following similarity measure given by [BMRB96] :

$$\sigma(l_u, l_s) = \frac{M(l_u \cap l_s)}{M(l_s)},$$

assuming the usual definition of the fuzzy intersection $\mu_{l_u \cap l_s} = min(\mu_{l_u}, \mu_{l_s})$ and the following fuzzy measure M :

$$M(l) = \begin{cases} \sum_{x \in D_A} \mu_l(x) & \text{if } D_A \text{ is finite} \\ \int_{D_A} \mu_l(x).dx & \text{otherwise.} \end{cases}$$

This choice is motivated by the fact that fuzzy set-based mapping operation can be given a semantics in terms of *degree of satisfiability* [BMRB96]. Indeed, $\sigma(l_u, l_s)$ evaluates the satisfiability of the user label l_u by a summary label l_s. Note that this measure is not symmetrical because l_u is taken as a reference. The behavior of $\sigma(l_u, l_s)$ is graphically illustrated on figure 2.12.

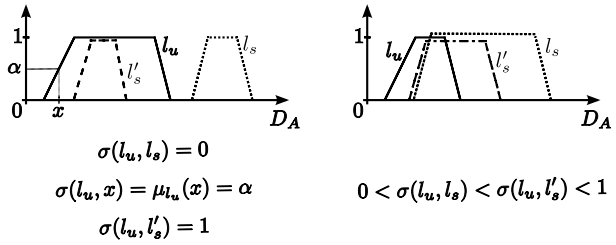


Figure 2.12 – The behavior of σ

Note that when $\tau = 0^+$ (respectively $\tau = 1$), we fall back into the Possibility (respectively Necessity) scenario. Note also that since crisp sets are special cases of fuzzy sets, this rewriting process can also be used to rewrite crisp queries (e.g., '`Price` $\in [150\text{k€}, 300\text{k€}]$').

Once the query is rewritten, *ESRA* could be performed in order to provide the summarized answer. The relevance of summaries in the result depends on the query mapping and especially the satisfiability degree of the rewritten predicates. Thus, we propose in the following an appropriate scoring function to sort *ESRA*'s results.

2.5.2 Results Sorting

Consider a query Q, expressed using the user's vocabulary. Further, let X be the set of attributes specified in Q, and let z be a summary returned by *ESRA* when performed on Q', the rewritten version of Q. We denote by Q_A and $z.A$ the set of linguistic labels that appear on attribute A in Q and z, respectively.

In the following, we define a scoring function $\mathsf{Score}(Q, z)$ that measures how well the summary z fits the user query Q. $\mathsf{Score}(Q, z)$ is based on the *satisfiability degrees* [BMRB96] of linguistic labels of Q by linguistic labels of z. More specifically, for each attribute $A \in X$, we first use $\sigma(l_u, l_s)$ to compute the satisfiability degree of each label $l_u \in Q_A$ by each label $l_s \in z.A$. Then, we combine all these pairwise degrees to compute $\sigma_G(Q_A, z.A)$, the global satisfiability degree of Q_A by $z.A$. The simplest way to do this is by using a disjunctive aggregation that aggregates values as an OR operator, so that the result of the combination is high if some (at least one) values are high. Disjunctive combination is typically performed in fuzzy set theory by T-conorm operators, whose most used instance is '$\max$'. Thus, we obtain :

$$\sigma_G(Q_A, z.A) = \max_{l_u \in Q_A, l_s \in z.A} \sigma(l_u, l_s)$$

The function σ_G establishes a comparison, attribute by attribute. Thus, we have as many values of comparison as there are attributes in X. A last aggregation step of those values is done using the product T-norm operator in order to obtain $\mathsf{Score}(Q, z)$:

$$\mathsf{Score}(Q, z) = \prod_{A \in X} \sigma_G(Q_A, z.A)$$

We use the product T-norm operator since it takes into account all σ_G values and balances the summary score value across each of the conditions in the query.

Thus, for each summary z from the set S of summaries returned by *ESRA*, we first compute its score $\mathsf{Score}(Q, z)$. Then, we sort these summaries by their scores in decreasing order and finally return the ordered list S_{sorted} to the user. Furthermore, when a user decides to explore a

summary z in the list S_{sorted} using the SHOWCAT option, we sort the set of z's children according to their scores in decreasing order. Note that all basic values $\sigma(l_u, l_s)$ needed are available because the intentional description of z includes the intentional description of its children. Likewise, when a user decides to explore z using SHOWTUPLES option, each tuple t can be sorted by $\mathsf{Score}(Q, t)$. In this case, the satisfiability degree $\sigma(l_u, t.A)$ of each label $l_u \in Q_A$ by each value $t.A$ of t on A is given by $\mu_{l_u}(t.A)$.

2.6 Experimental Results

Evaluating and comparing the effectiveness of different approaches that address the *Many-Answers* problem in databases is challenging. Unlike Information Retrieval where there exist extensive user studies and available benchmarks (e.g., the TREC[23] collection), such infrastructures are not available today in the context of Relational Databases. Nonetheless, in this section, we discuss the efficiency and the effectiveness of *ESA*, *ESA+SEQ* and *ESRA* algorithms based on a real database.

2.6.1 Data Set

Through an agreement, the CIC Banking Group provided us with an excerpt of statistical data used for behavioral studies of customers. The dataset is a collection of 33735 customers defined on 10 attributes (e.g., age, income, occupation, etc.). On each attribute, marketing experts defined between 3 and 8 linguistic labels leading to a total of 1036800 possible label combinations (i.e., 1036800 possible cells). Note that all the experiments were done on a 1.7GHz P4-based computer with 768MB memory.

2.6.2 Results

All experiments reported in this section were conducted on a workload composed of 150 queries with a random number of selection predicates from all attributes (i.e., each query has between 1 and 3 required features on $1, 2, 3$ or 4 attributes).

[23] http ://trec.nist.gov

Quantitative Analysis

The CIC dataset is summarized by the SAINTETIQ system as described in Section 2.1. The dataset, consisting of 33735 records, yields a summary tree with 13263 nodes, 6701 leaves or cells, maximum depth of 16, average depth of 10.177, maximum width of 14 and an average width of 2.921. The data distribution in the summary tree reveals a 0.6% ($\frac{6701}{1036800}$) occupation rate.

From the analysis of theoretical complexities, we claim that *ESA* and *ESRA* are much faster than the post-clustering approach *ESA+SEQ*. That is the main result of Figure 2.13 that shows the performance evolution according to the number of cells populated by query answer records. Furthermore, we plot the number of summary nodes visited (#Visited Nodes) per query (right scale) and finally, the normalized *ESRA* time cost ($t_{\text{N. ESRA}}$) to evaluate the performance of *ESRA* regardless of how the query fits the pre-clustering summary hierarchy. $t_{\text{N. ESRA}}$ is computed as follows :

$$t_{\text{N. ESRA}} = \frac{t_{\text{ESRA}} \cdot \#\text{Tree Nodes}}{\#\text{Visited Nodes}}.$$

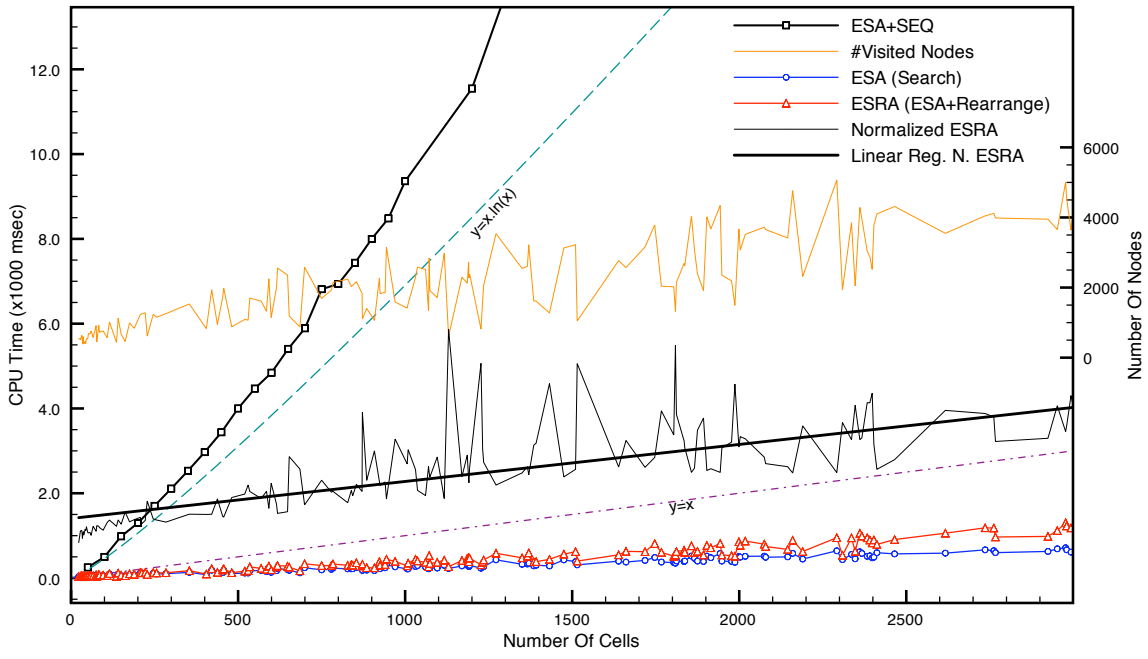


Figure 2.13 – Time cost comparison

As one can observe, Figure 2.13 verifies experimentally that *ESA+SEQ* is quasi-linear ($O(L \cdot \log L)$) in the number of cells L whereas *ESA*, *ESRA* and *N. ESRA* are linear ($O(L)$). Besides, the time cost incurred by rearranging query results (i.e., $t_{\text{ESRA}} - t_{\text{ESA}}$) is insignificant compared

to the search cost (i.e., t_{ESA}). For instance, for $L = 1006$, t_{ESA} and t_{ESRA} are 0.235sec and 0.287sec, respectively. Thus, the *ESRA* algorithm is able to drastically reduce the time cost of clustering query results.

Qualitative Analysis

Due to the difficulty of conducting a large-scale[24] real-life user study, we discuss the effectiveness of the *ESRA* algorithm based on structural properties of results provided to the end-user. It is worth noticing that the end-user benefit is proportional to the number of items (clusters or tuples) the user needs to examine at any time as well as to the dissimilarity between these items.

We define the 'Compression Rate' (CR) as the ratio of the number of clusters (summaries) returned as a starting point for an online exploration over the total number of cells covered by such summaries. Note that $CR = 1$ means no compression at all, whereas smaller values represent higher compression. As expected, Figure 2.14 shows that CR values of *ESRA* and *ESA+SEQ* are quite similar and much smaller than that of *ESA*. Thus, size reduction is clearly achieved by the *ESRA* algorithm.

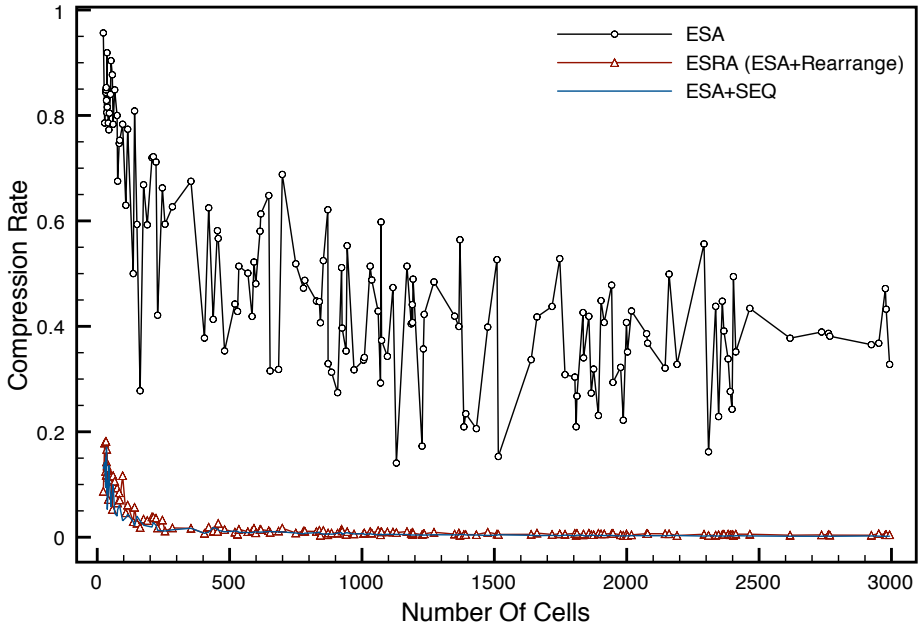


Figure 2.14 – Compression rate comparison

We could also see that the dissimilarity (Figure 2.15) of the first partitioning that *ESRA*

[24] A potential problem with real-user evaluation techniques is that users' opinions are very subjective. Hence, even if we obtain positive feedback from a small set of test users, we cannot be more convinced and affirmative about the effectiveness of our approach.

presents to the end-user is greater than that of *ESA* and is in the same order of magnitude than that provided by *ESA+SEQ*. It means that *ESRA* significantly improves discrimination between items when compared against *ESA* and is as effective as the post-clustering approach *ESA+SEQ*. Furthermore, the dissimilarity of the *ESA* result is quite similar to that of the most specialized partitioning , i.e., the set of cells. Thus the rearranging process is highly required to provide the end-user with well-founded clusters.

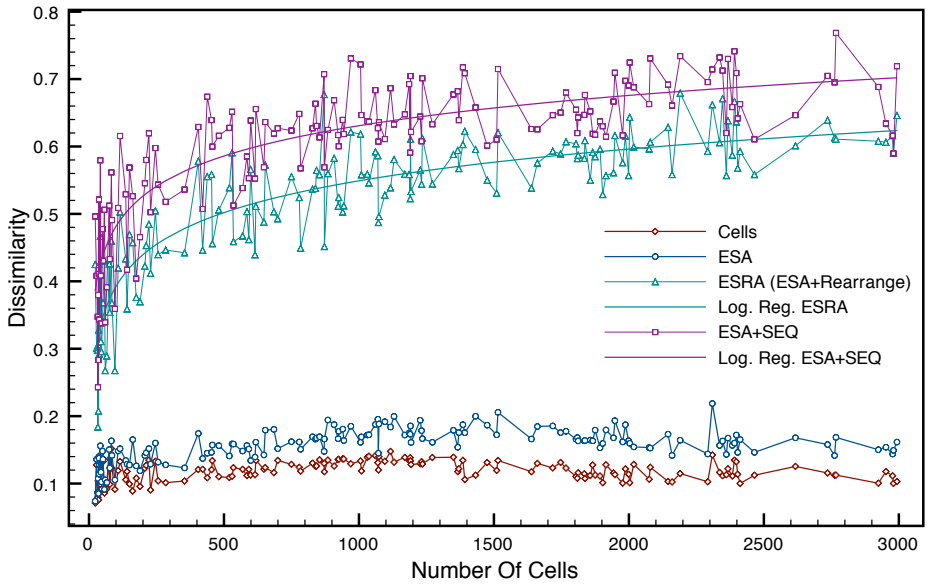


Figure 2.15 – Dissimilarity comparison

Now assume that the user decides to explore a summary z returned by *ESRA*. We want to examine the number of hops NoH (i.e., the number of SHOWTUPLES/SHOWCAT operations) the user might employ to reach a relevant tuple. NoH ranges from 1 up to $d_z + 1$, where d_z is the height of the tree rooted by z (i.e., subtree H_z). The best case (NoH $= 1$) occurs when z exactly fits the user's need, whereas the worst case (NoH $= d_z + 1$) occurs when the relevant information is reached by following a path of maximal length in H_z. Note that these two scenarios are on opposite extremes of the spectrum of possible situations : the generalized case ($0 \leq$ NoH $\leq d_z + 1$) is that the user's need is successfully served by a node at height h such that $0 \leq h \leq d_z$.

In the following experiment (Figure 2.16), one considers a user query q with selectivity of η, i.e., the number of tuples returned by q divided by the total number of tuples in the database (33735). We look for all the possible summaries (subtrees) in the pre-computed summary hierarchy that can be returned as the result of q and, for each one, we compute its maximum depth, its

average depth and the average length of all paths emanating from that node (summary). Then, we pick out the highest (maximum) value observed for each of these measures. The three values obtained, for each value of η ($\eta \in \{0.2\%, 0.4\% \ldots 2\%\}$), evaluate respectively :

(1) the worst number of hops required to reach the deepest leaf node (Worst NoH2C) containing relevant data ;

(2) the average number of hops needed to reach any leaf node (Average NoH2C) containing relevant data (i.e., not necessarily the deepest one) ;

(3) the average number of hops required to reach any node (Average NoH2N) containing relevant data (i.e., not necessarily a leaf node).

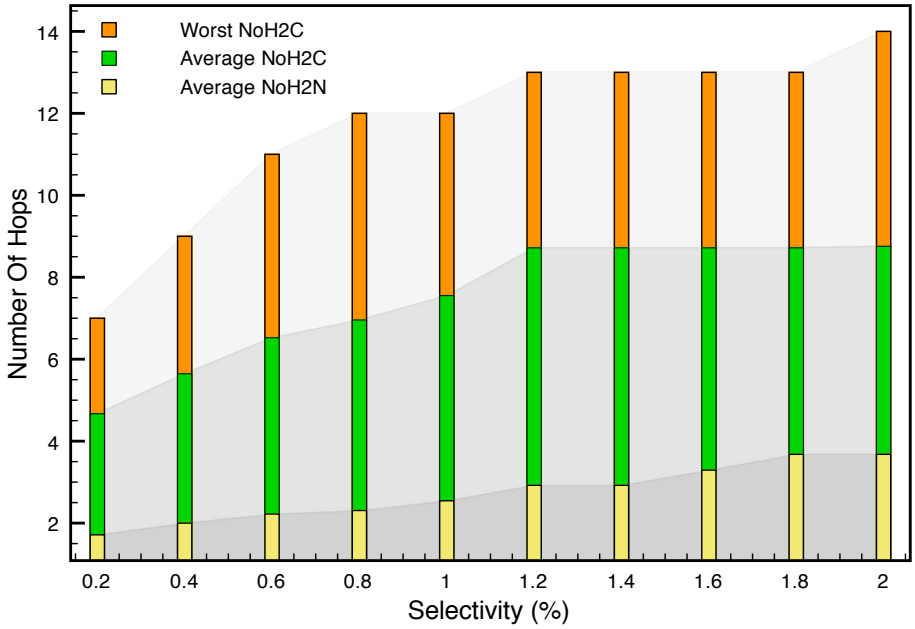


Figure 2.16 – The number of SHOWTUPLES/SHOWCAT operations

Figure 2.16 shows that the Worst NoH2C and the Average NoH2C are relatively high, but bounded respectively by the maximum (16) and the average (10.177) depth of the pre-computed tree. It is worth noticing that, in real life situation, the user finds out that her/his need has been successfully served by an inner node of the tree rooted by z. Thus, the Average NoH2N is more adapted to evaluating the effectiveness of our approach (*ESRA*). As one can observe, the Average NoH2N is quite small given the number of tuples in the q result set. For instance, NoH2N $= 3.68$ is the number of hops the user takes to reach relevant information within a set of 674 tuples ($\eta = 2$).

Finally, we aim to compare the number of results the user has to examine until she/he reaches relevant data when answers are rearranged using *ESRA* or not. To this end, we pick first randomly one summary z from the pre-computed tree such that z covers 30 cells. Then, we generate the query q that returns z. In this experiment, we assume that z exactly fits the user's need, i.e., the user is interested in all tuples (R_z) that match q. Further, we assume that user submits a broad query q' instead of q, i.e., q' is a relaxation of q and consequently matches a superset of R_z. Note that when no processing has been performed on q' results, the user has to examine N cells and for each one perform one SHOWTUPLES operation in order to get all relevant tuples (R_z), where N is the number of cells that satisfy q'. However, when q' is processed using the *ESRA* algorithm, the user has to examine only ENoH clusters and then perform one SHOWTUPLES operation ; ENoH is the effective number of hops (or SHOWCAT operations) required to reach z. This is because the summary z exactly matches the query q' (relaxation of q), and consequently it is returned as a whole by *ESRA*. Figure 2.17 shows the evolution of $\frac{\text{ENoH}}{N}$ according to the precision of q', i.e, the number of relevant cells (i.e., 30) divided by the number of cells that match the query q' (i.e., N). As one can observe, $\frac{\text{ENoH}}{N}$ is much smaller than 1. For instance, it is equal to 0.0013 ($\frac{5}{3603}$) for a query precision of 0.83% ($\frac{30}{3603}\%$). It means that only 5 hops are required to locate the summary z within a clustering schema of q' results that covers 3603 cells. Thus, *ESRA* is able to drastically reduce the 'Information Overhead' that users experience when submitting a broad query.

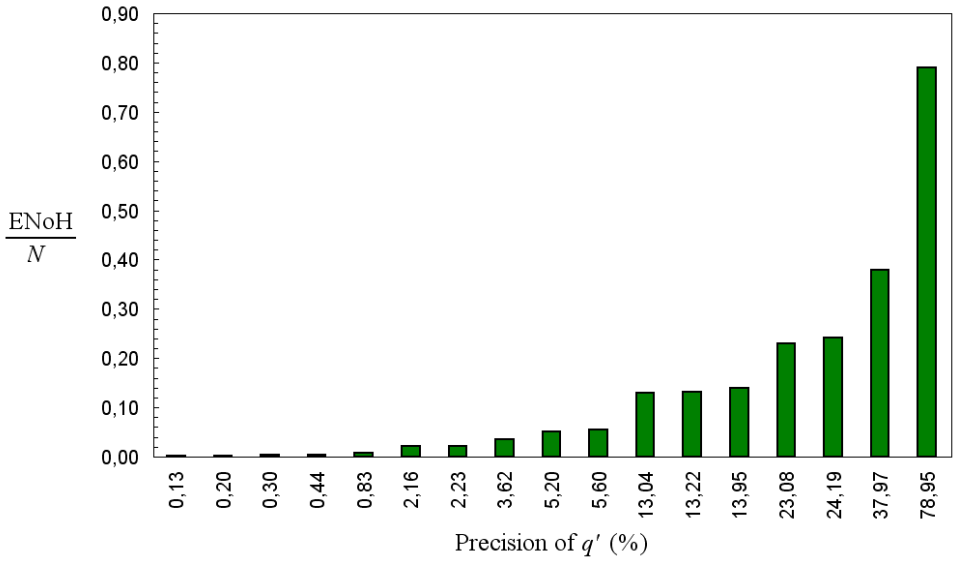


Figure 2.17 – Effectiveness of *ESRA* regarding the precision of the user query

Those experimental results validate the claim of this work, that is to say the *ESRA* algorithm

is very efficient (Figure 2.13) and provides useful clusters of query results (Figure 2.14 and Figure 2.15) and consequently, makes the exploration process more effective (Figure 2.16 and Figure 2.17).

2.7 Conclusion

Interactive and exploratory data retrieval are more and more suitable to database systems. Indeed, regular 'blind' queries often retrieve too many answers. Users then need to spend time sifting and sorting through this information to find relevant data.

In this chapter, we proposed an efficient and effective algorithm coined Explore-Select-Rearrange Algorithm (*ESRA*) that uses database SAINTETIQ summaries to quickly provide users with concise, useful and structured representations of their query results. Given a user query, *ESRA* (i) explores the summary hierarchy (computed offline using SAINTETIQ) of the whole data stored in the database ; (ii) selects the most relevant summaries to that query ; (iii) rearranges them in a hierarchical structure based on the structure of the pre-computed summary hierarchy and (iv) returns the resulting hierarchy to the user. Each node (or summary) of the resulting hierarchy describes a subset of the result set in a user-friendly form using linguistic labels. The user then navigates through this hierarchy structure in a top-down fashion, exploring the summaries of interest while ignoring the rest.

Experimental results showed that the *ESRA* algorithm is efficient and provides well-formed (tight and clearly separated) and well-organized clusters of query results. Thus, it is very helpful to users who have vague and poorly defined retrieval goals or are interested in browsing through a set of items to explore what choices are available.

The *ESRA* algorithm assumes that the summary hierarchy of the queried data is already built using SAINTETIQ and available as input. However, the SAINTETIQ model requires full access to the data which is going to be summarized, i.e., all data has to be located at the site where it is summarized. This requirement severely limits the applicability of the *ESRA* algorithm in a distributed environment, where data are distributed across many sites and their centralization cannot scale in terms of communication, storage and computation. In the next chapter, we study the behaviors and properties of the SAINTETIQ summarization technique in order to extend its ability to deal with distributed databases.

CHAPTER 3

Merging Distributed Database Summaries

Introduction

Nowadays, data are often stored in different databases located on distant sites (e.g., PCs connected to the Internet) rather than being stored in a centralized database. This is due to several reasons, including the following :

- many organizations and companies are scattered worldwide geographically. Each site generates its own data and manages its own data repository. For instance, supermarket chains such as Carrefour, Monoprix etc. have some data which is located in Paris and some data in Nantes ;
- computer systems have limits. These limitations can be seen in the amount of memory the system can address, the number of hard disk drives which can be connected to it or the number of processors it can run in parallel. In practice this means that, as the quantity of information in a centralized database becomes larger, a single system can no longer cope with all the information that needs to be stored and queried.

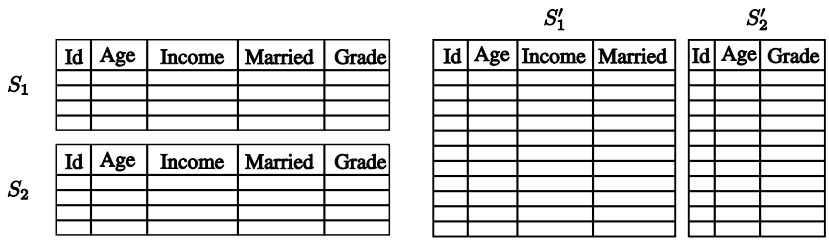


Figure 3.1 – (a) Horizontally fragmented data (b) Vertically fragmented data

In a distributed environment, data may be horizontally or vertically partitioned (fragmented [OV99]) among different sites. Figure 3.1 illustrates these two cases. In case of horizontal partitioning, the feature space is the same for the different databases. Essentially this boils down

to different data sites collecting the same kind of information over different (possibly overlapping) groups of individuals, e.g., site S_1 and site S_2 in Figure 3.1. On the other hand, in case of vertical partitioning, different databases define different feature spaces (with a set of possibly overlapping attributes). Basically this means that data is collected by different sites on the same individuals but with different feature sets, e.g., site S'_1 and site S'_2 in Figure 3.1.

Due to some concerns related to confidentiality, storage, communication bandwidth and/or power limitation, the data cannot be transmitted to a central site. Consequently, the current centralized version of SAINTETIQ (Section 2.1, Chapter 2) is either not desirable (privacy reasons) or not feasible (resource limitations). For instance, in medical databases, only anonymous and statistical information are available since individual information (such as the name, the address and the phone number) may violate the patient confidentiality. Even if privacy is not an obstacle, transmitting the entire local data set to a central site and performing the clustering are, in some application areas, quite difficult, if not almost impossible. In astronomy, for instance, data sets gathered by telescopes and satellites spread all over the world, are measured in gigabytes and even terabytes.

To enable summarization of distributed data without a prior "unification" of the data sources, we propose new algorithms that take two summary hierarchies (local models) as input, instead of the raw data, and merge them to output a global summary hierarchy of the entire distributed data :

- *Union by Incorporation Algorithm* (*UIA*) and *Union by Alignment Algorithm* (*UAA*) to deal with horizontally distributed data. They rely on a *summary union* operator which aggregates two input summaries (or clusters) representing two different sets of database records with the same set of attributes ;
- *Subspace-Oriented Join Algorithm* (*SOJA*) and *Tree Alignement-based Join Algorithm* (*TAJA*) to deal with vertically distributed data. They rely on a *summary join* operator which appends two input summaries (or clusters) representing the same set of database records but with two different feature sets.

The main concern of this chapter is to introduce and compare those algorithms and study the pros and cons of each one. This raises some questions :

- How can we define the set of the best representatives (or summaries) of a given data set according to the input summaries ?
- What are the main criteria (time complexity, model consistency, etc.) that need to be taken care of in order to ensure that our merging algorithms (*UIA*, *UAA*, *SOJA* and *TAJA*)

overcome the SAINTETIQ limitations in distributed systems ?

- How good are the merged (global) hierarchies ?

The rest of this chapter is organized as follows. In the next section, we discuss in detail the problem of summarizing distributed data. Our merging algorithms (*UIA*, *UAA*, *SOJA* and *TAJA*) are presented in Section 3.2. Section 3.3 introduces how to evaluate the correctness of merging algorithm results using an appropriate and consistent clustering validity index. Moreover, an experimental study is presented in Section 3.4. In Section 3.5, we briefly discuss some works which are related to the issue addressed in this chapter. Section 3.6 concludes.

3.1 Problem Analysis

In this section, we define the problem of summarizing distributed data and present a toy example that will be used throughout this chapter. Then, we describe some basic approaches to address such problem and point out their limitations.

3.1.1 Problem Statement

Let $E = \{A_1, \ldots, A_N\}$ be a set of attributes, and let $\{X, Y, Z\}$ be a partition of E. Assume that the N-dimensional space $A_1 \times A_2 \times \ldots \times A_N$ is equipped with a grid that defines basic N-dimensional areas called cells in E. The cells are obtained, for example, by partitioning every initial feature domain into several sub-domains using linguistic labels from a *Knowledge Base (KB)*. Assume also that there exist four relational database tables R_1, R_2, R'_1 and R'_2 located respectively on distant sites S_1, S_2, S'_1 and S'_2. R_1 and R_2 are defined on E, i.e., R_1 and R_2 are horizontal fragments of the global relation $R_1 \cup R_2$. R'_1 and R'_2 are defined respectively on different feature sets $E_1 = X \cup Y$ and $E_2 = Y \cup Z$. Furthermore, we assume that there is a common feature ID, accessible to S'_1 and S'_2, that can be used to associate a given sub-tuple in site S'_1 to a corresponding sub-tuple in site S'_2. In other words, R'_1 and R'_2 are vertical fragments of the global relation $R'_1 \bowtie R'_2$[25]. Note that the latter assumption is required for a reasonable solution to the problem of summarizing distributed data and is not overly restrictive. Indeed, even if there is no such ID, any entity resolution method [BGL06, SD06] could provide the association. Moreover, we assume consistency [BK99, SK05] between values of attributes that appear in R'_1 and R'_2, i.e., if a tuple with ID id_1 has A-value v in R'_1 then we assume that tuple with ID id_1 in R'_2 will also have value v for its A attribute.

[25]The natural join of R'_1 and R'_2 over the common feature ID (i.e., $R'_1 \bowtie_{\mathsf{ID}} R'_2$)

Definition 3.1. ***Problem Definition.***
We define the problem of summarizing horizontally (resp., vertically) distributed data as follows : compute $H_{R_1 \cup R_2}$ (resp., $H_{R'_1 \bowtie R'_2}$), the summary hierarchy of data located at S_1 and S_2 (resp., S'_1 and S'_2).

The traditional solution to the above problem is to simply transfer R_1 and R_2 (resp., R'_1 and R'_2) to a central site where their union (resp., join) is performed, and then the summary hierarchy of distributed data is computed by applying SAINTETIQ over $R_1 \cup R_2$ (resp., $R'_1 \bowtie R'_2$). Such an approach, however, is either infeasible or undesirable for several reasons :

- Storage cost : the central storage system needs to be very large in order to host all data from the distributed sites.
- Computational cost : the central system has to be very powerful in order to deal with huge data volumes.
- Communication cost : moving distributed data to a central location might take an extremely long time and require huge network bandwidth.
- Private and sensitive data : moving raw data to the central site is not desirable since it puts privacy at risk.

On the other hand, it is possible to summarize the data locally where it has been generated and stored. The resulting summary hierarchies (local models) can then be sent to a central site where they are *merged* to obtain the summary hierarchy (global model) of the entire distributed data. This approach is very attractive because : (1) local models can be computed quickly and independently from each other ; (2) local models are much smaller than raw data and sending them, instead of raw data, reduces transmission cost ; (3) sharing only local models, instead of raw data, would give reasonable security since it overcomes partially the issues of privacy and security of raw data. We therefore discuss new algorithms for *merging* local models.

3.1.2 Running Example

To illustrate our proposal, we introduce here a sample data set $R(\mathsf{ID}, A, B, C)$ with 30 records represented on four attributes : ID, A, B and C. We suppose that $\{a_1, a_2, a_3\}$, $\{b_1, b_2, b_3\}$ and $\{c_1, c_2\}$ are sets of linguistic labels defined respectively on the attributes A, B and C. Figure 3.2 shows the summary hierarchy H_R provided by SAINTETIQ performed on the data set R.

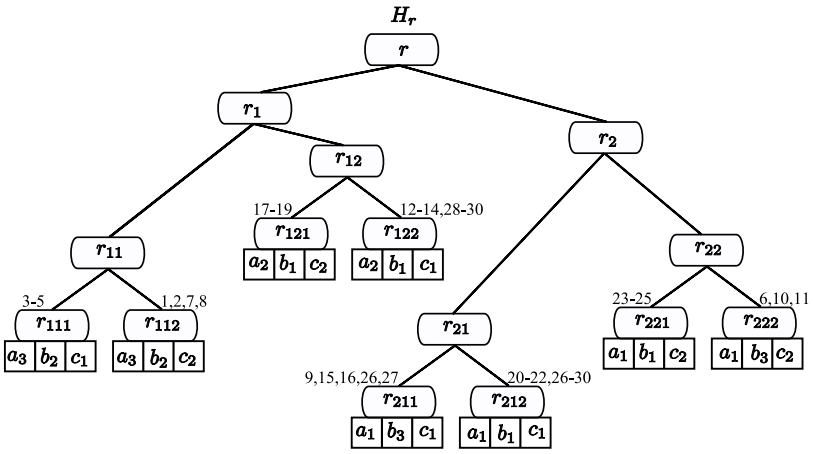


Figure 3.2 – Summary hierarchy H_R of the data set R

To serve merging processes, we first perform a horizontal (resp., vertical) fragmentation of R : we then obtain two relations $R_1(\mathsf{ID}, A, B, C)$ and $R_2(\mathsf{ID}, A, B, C)$ (resp., $R'_1(\mathsf{ID}, A, B)$ and $R'_2(\mathsf{ID}, B, C)$). Then we also apply SAINTETIQ on these four derived relations to produce the corresponding summary hierarchies (Figure 3.3 and Figure 3.4) that should be merged.

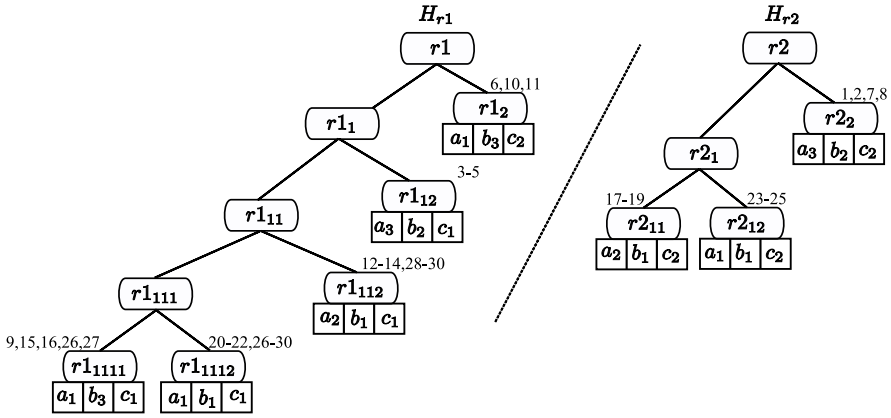


Figure 3.3 – H_{R_1} and H_{R_2}

3.1.3 Basic Approaches

In this section, we discuss first ideas for *merging* two summary hierarchies and describe their drawbacks. But before proceeding further, we define new operators that will be used in the following.

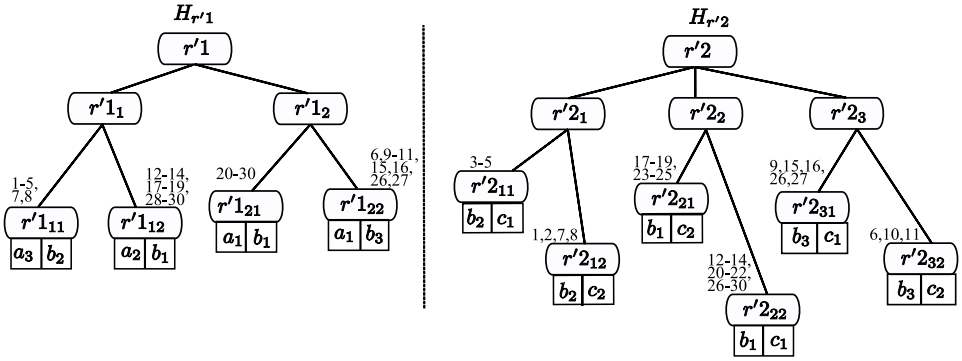


Figure 3.4 – $H_{R'_1}$ and $H_{R'_2}$

3.1.3.1 Merging Operators

Definition 3.2. ***Summary Union Operator*** **($\widetilde{\cup}$).**
Let z_1 and z_2 be two summaries (Definition 2.1, Chapter 2) of $E = \{A_1, \ldots, A_N\}$, i.e., z_1 and z_2 are areas of E. We define a union operator for z_1 and z_2 as :

$$z^\star = z_1 \widetilde{\cup} z_2 \iff \begin{array}{l} L_{z^\star} = L_{z_1} \cup L_{z_2} \\ R_{z^\star} = R_{z_1} \cup R_{z_2} \end{array}$$

Recall that, for a given summary z, L_z denotes the set of cells covered by z, R_z denotes its extent, and $\langle z.A_1, \ldots, z.A_N \rangle$ (or simply $\langle z.E \rangle$) denotes its intent ($z.A_i$ is the set of linguistic labels that appear in z on the attribute A_i).

Figure 3.5 represents the summary resulting from the union of the summaries $r1_{111}$ and $r2_{11}$ of H_{R_1} and H_{R_2} (Figure 3.3), respectively.

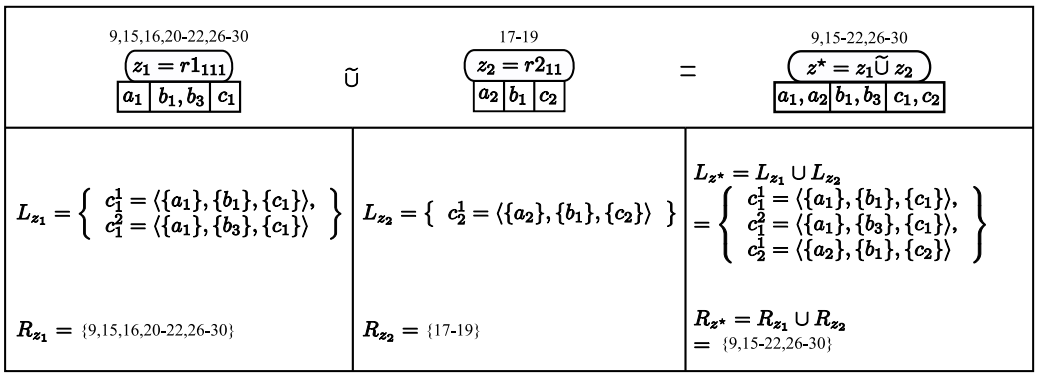


Figure 3.5 – $z^\star = r1_{111} \widetilde{\cup} r2_{11}$

Definition 3.3. ***Summary Join Operator ($\widetilde{\bowtie}$).***
Let $\{X, Y, Z\}$ be a partition of $E = \{A_1, \ldots, A_N\}$, and let z_1 and z_2 be respectively summaries of $E_1 = \{X, Y\}$ and $E_2 = \{Y, Z\}$, i.e., z_1 is an area of E_1 and z_2 is an area of E_2. We define a join operator for z_1 and z_2 as :

$$z^\star = z_1 \widetilde{\bowtie} z_2 \iff \begin{cases} L_{z^\star} = \emptyset & \text{if } z_1 \text{ and } z_2 \text{ are both cells and } \langle z_1.Y\rangle \neq \langle z_2.Y\rangle, \\ L_{z^\star} = \{\langle z_1.X, z_1.Y, z_2.Z\rangle\} & \text{if } z_1 \text{ and } z_2 \text{ are both cells,} \\ L_{z^\star} = \bigcup_{c_1 \in L_{z_1}, c_2 \in L_{z_2}} L_{c_1 \widetilde{\bowtie} c_2} & \text{otherwise.} \end{cases}$$

$$R_{z^\star} = R_{z_1} \bowtie R_{z_2}$$

In the above definition, for a given summary z_1 and a given set of attributes $Y = \{Y_1, \ldots, Y_m\} \subseteq E$, $\langle z_1.Y\rangle$ is used to denote $\langle z_1.Y_1, \ldots, z_1.Y_m\rangle$. Furthermore, $z_1.Y \neq z_2.Y$ is a compact notation for " $\exists\, Y_i \in Y$ such that $z_1.Y_i \neq z_2.Y_i$ ".

$z^\star = z_1 \widetilde{\bowtie} z_2$ is a summary of $E = E_1 \cup E_2 = \{X, Y, Z\}$, i.e., $z^\star$ is an area of E. It is worth noticing that $z^\star$ could be empty as soon as $R_{z^\star} = \emptyset$ or $L_{z^\star} = \emptyset$.

Figure 3.6 represents the summary resulting from the join of the summaries $r'1_2$ and $r'2_2$ of $H_{R'_1}$ and $H_{R'_2}$ (Figure 3.4), respectively.

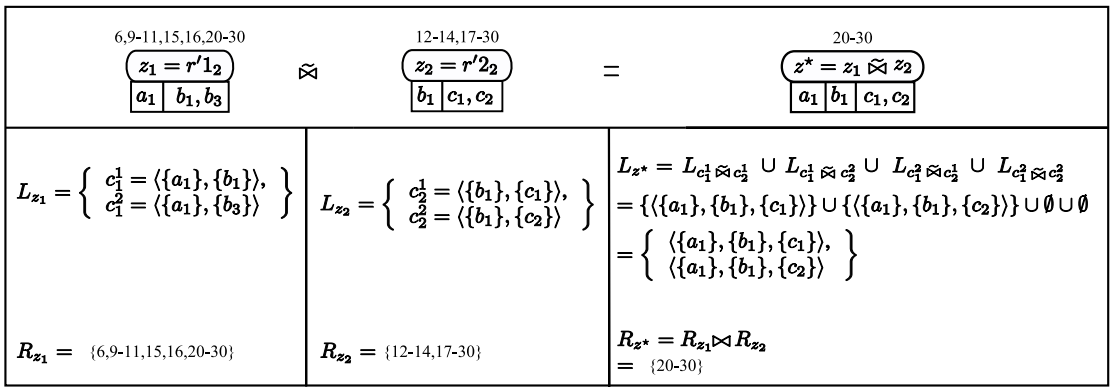


Figure 3.6 – $z^\star = r'1_2 \widetilde{\bowtie} r'2_2$

Definition 3.4. ***Partition Union Operator*** **($\widehat{\cup}$).**
Let R_1 and R_2 be two relations defined on $E = \{A_1, \ldots, A_N\}$, and let P_1 and P_2 be respectively summary partitioning (Definition 2.3, Chapter 2) of R_1 and R_2. The union of P_1 and P_2 is defined as the set of summaries $\Pi = P_1 \widehat{\cup} P_2$ which verifies the following conditions :

(1) $\forall z \in \Pi, \exists\ F \subseteq P_1 \cup P_2$ such that $z = \widetilde{\cup}_{z' \in F}\ z'$, i.e., z is either an element of $P_1 \cup P_2$ or a union of elements of $P_1 \cup P_2$;

(2) $\forall z, z' \in \Pi$, z and z' are disjoint (Property 2.1, Chapter 2) ;

(3) for each $\Pi' \neq \Pi$ that verifies (1) and (2), $|\Pi| > |\Pi'|$.

Recall that $|.|$ denotes the cardinality of a set. Considering the above definition, if all summaries of $P_1 \cup P_2$ are pairwise disjoint (resp., overlapping), then $\Pi = P_1 \cup P_2$ (resp., $\Pi = \{\widetilde{\cup}_{z \in P_1 \cup P_2}\ z\}$). Note that $P_1 \widehat{\cup} P_2$ is trivially a summary partitioning of $R_1 \cup R_2$.

Definition 3.5. ***Partition Join Operator*** **($\widehat{\bowtie}$).**
Let $\{X, Y, Z\}$ be a partition of $E = \{A_1, \ldots, A_N\}$, R'_1 and R'_2 two relations defined respectively on $E_1 = X \cup Y$ and $E_2 = Y \cup Z$, P_1 a summary partitioning of R'_1 and P_2 a summary partitioning of R'_2. The join of P_1 and P_2 is defined as :

$$P_1 \widehat{\bowtie} P_2 = \left\{ z_1 \widetilde{\bowtie} z_2 \mid (z_1 \in P_1) \wedge (z_2 \in P_2) \right\}$$

$P_1 \widehat{\bowtie} P_2$ is a summary partitioning of $R'_1 \bowtie R'_2$. Indeed, we can easily check that $P_1 \widehat{\bowtie} P_2$ verifies the disjunction property (Property 2.1, Chapter 2) and the coverage property (Property 2.2, Chapter 2). Hereafter, we shall use $P_1 \widehat{\ltimes} P_2$ to denote the set of summaries of P_1 such that :

$$P_1 \widehat{\ltimes} P_2 = \{ z_1 \in P_1 |\ (\exists\ z_2 \in P_2)\ \wedge\ (z_1 \widetilde{\bowtie} z_2 \in P_1 \widehat{\bowtie} P_2) \}$$

Recall that the most specialized summary partitioning of a given relation R is the set of cells L_R that are populated by records of R. Thus from Definition 3.4 and Definition 3.5, we have the following equalities :

$$L_{R_1 \cup R_2} = L_{R_1} \widehat{\cup} L_{R_2}$$

$$L_{R'_1 \bowtie R'_2} = L_{R'_1} \widehat{\bowtie} L_{R'_2}$$

It means that the most specialized summary partitioning of distributed data can be obtained directly from those of local data, and therefore, no mapping process is needed (Section 2.1.1.1, Chapter 2).

It is worth noticing that the exact values of attribute-dependent (e.g., mean, maximum, minimum and standard deviation) and attribute-independent (e.g., record count) statistical information of each cell $c^\star$, a cell resulting from the union or join of two initial cells c_1 and c_2, cannot be computed without either centralization of data or communication between sites. Indeed, such parameters are calculated directly from data. Nevertheless, if univariate-distribution laws (e.g., normal, uniform, etc.) of attributes values in c_1 and c_2 are known, an approximation of $c^\star$'s attribute-dependent measures can be calculated. An approximation of $c^\star$'s attribute-independent parameters is a bit more complicated, but not impossible. First, the multivariate-distribution law is approximated using the univariate-distribution laws. Then, it could be used to estimate $c^\star$'s attribute-independent measures.

3.1.3.2 The Greedy Merging Algorithm (*GMA*)

The *Greedy Merging Algorithm* (*GMA*) is the straightforward way of summarizing distributed data. It operates exhaustively on all the populated cells to find out the best partitioning of the global data set. Indeed, depending on the assumed case - horizontally or vertically distributed data - *GMA* takes the union or the join of the sets of leaves (cells) of the input hierarchies and generates their *optimal* partitioning schema. We define here the optimality as the result of performing the following algorithm on a set of summaries $\mathcal{Z}$:

1. compute the partition lattice $\mathcal{L}$ of $\mathcal{Z}$;
2. for each partition $P \in \mathcal{L}$, build summary descriptions (hyperrectangles) from clusters of cells ;
3. filter from $\mathcal{L}$ the set of candidate summary partitions $\mathbf{P}$ that are partitions satisfying the disjunction property (coverage is trivial) ;
4. the optimal partitioning of $\mathcal{Z}$ is the partition $P_{best} \in \mathbf{P}$ with the highest U value[26].

Note that at the fourth stage, we can also compute the summary utility of the worst partitioning P_{worst} of $\mathcal{Z}$. We will use such a partition for the evaluation of our algorithms (Section 3.3).

From the optimal summary partitioning P_{best}, a hierarchical organization of summaries is built by agglomerating and dividing summaries to provide higher and lower nested partitions such that the U function may emphasize local or global patterns in the data. Thus, we obtain two different hierarchies according to the performed optimization (local or global). None of them is preferred to the other and the third-party application or user's requirements are key factors to selecting the right optimization mode. Indeed, the former based on local optimization seems

[26]Recall that U denotes the objective function of SAINTETIQ.

more appropriate for applications including querying, analyzing and browsing operations. It supports fruitful strategies to explore large information space. The latter is a more relevant choice for applications that address the problem of resource limitation to substitute an entire summary partition to the original data set.

The two optimization approaches above (i.e., local or global) are described in detail in the following subsections where, for convenience, we denote by op either $\cup$ or $\bowtie$, and we denote by $F_1 \ \mathsf{op} \ F_2$ either the union of two horizontal fragments F_1 and F_2 or the join of two vertical fragments F_1 and F_2.

Global Optimization-based GMA

The main idea of this approach is rather simple. It starts from a given summary partition and recursively searches in two directions for the highest U-valuated partitions that are nested or including. It stops when it reaches respectively the cells (the leaves) and a single hyperrectangle (the root $z^\star$).

Recall that we are trying to build $H_F = H_{F_1 \ \mathsf{op} \ F_2}$, the summary hierarchy of relation $F = F_1 \ \mathsf{op} \ F_2$, from H_{F_1} and H_{F_2}. The global optimization-based GMA (*GO-GMA*) is performed on the optimal partition P_{best} of the set of leaves $L_F = L_{F_1} \ \widehat{\mathsf{op}} \ L_{F_2}$ as follows :

1. $l = 0$; $P_l \leftarrow P_{best}$;
2. While $P_l \neq \{z^\star\}$ do
   - generate the list G_l of summary partitions that generalize P_l ;
   - compute scores $\mathsf{U}(P)$ of every $P \in G_l$; rank P in G_l w.r.t. $\mathsf{U}(P)$;
   - $P_{l+1} \leftarrow top(G_l)$; l++ ;
3. return the nested partitions $P_k, k \in [0..l]$ as upper levels of the summary hierarchy H_F.

Function $top(G_l)$ returns the first item in the list, that is the summary partition with the highest U value. The algorithm above builds the upper part of the summary hierarchy H_F, from the root $P_l = \{z^\star\}$ to the best partitioning $P_0 = P_{best}$. The lower part is similarly computed, with the stop condition $P_l = L_F$ and G_l is S_l, the list of summary partitions that *specialize* P_l.

Figure 3.7 illustrates the merged hierarchy obtained from summary hierarchies shown in Figure 3.3 according to the *GO-GMA* approach. Note that it is the same as the one provided by *GO-GMA* when performed on the summary hierarchies $H_{R'_1}$ and $H_{R'_2}$ shown in Figure 3.4 since $R_1 \cup R_2 = R'_1 \bowtie R'_2$; and therefore $L_{R_1} \widehat{\cup} L_{R_2} = L_{R'_1} \widehat{\bowtie} L_{R'_2}$.

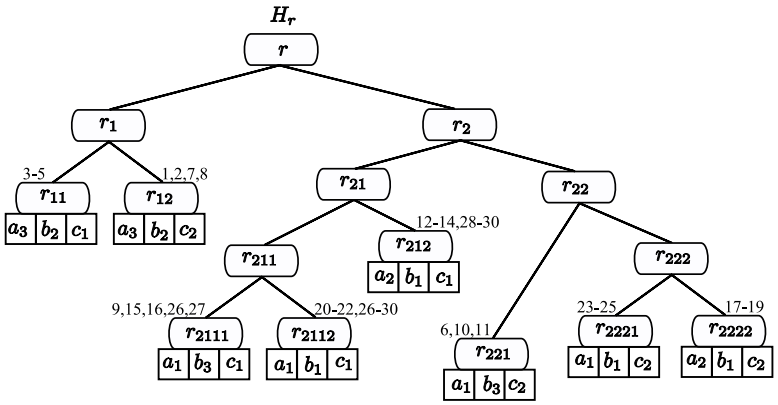


Figure 3.7 – *GO-GMA* on H_{R_1} and H_{R_2}

Local Optimization-based GMA

This approach takes the set of leaves $L_F = L_{F_1}\ \widehat{\mathsf{op}}\ L_{F_2}$ and computes the highest U-valuated partition P_{best} of L_F as the first level of the global hierarchy H_F. Then, it recursively applies to every node of P_{best}. The method uses a local optimization since each summary is subject to an individual clustering, from the root to the leaves.

Thus, the local optimization-based GMA (*LO-GMA*) computes the hierarchy H_F from H_{F_1} and H_{F_2} as follows :

1. search for P_{best} on L_F ; P_{best} is then the top-level partition in the tree H_F and is connected to the root $z^\star$;
2. for each $z \in P_{best}$ do
   - nothing if z is a leaf, else
   - repeat from step 1 with $L_F \leftarrow L_z$; $H_F \leftarrow H_{R_z}$; $z^\star \leftarrow z$;

Recall that L_z denotes the set of cells covered by the hyperrectangle z and R_z denotes the extent of z.

Figure 3.8 gives the result hierarchy of the summary hierarchies H_{R_1} and H_{R_2} shown in Figure 3.3 according to the *LO-GMA* approach.

Discussion

Both *GO-GMA* and *LO-GMA* are expensive since they rely on finding the best partition P_{best} of various data sets with nesting constraints. In fact, this principle requires to explore all the possible partitions each time. More specifically, the time complexity T_{GMA} of both *GO-GMA*

and *LO-GMA* approaches verifies :

$$T_{GMA}(L) = k_{OMA} \cdot \mathcal{B}_L \in O(2^L)$$

where L is the number of cells populated by records of the global data set, k_{OMA} a constant factor and $\mathcal{B}$ the Lth *Bell number*[27].

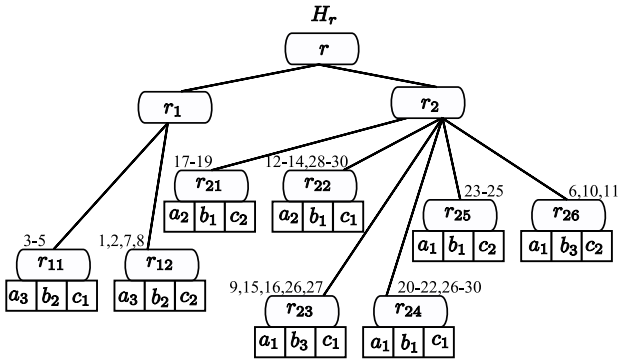


Figure 3.8 – LO-GMA on H_{R_1} and H_{R_2}

Nevertheless, in the computation of the optimal partitioning, it is more efficient to perform the checking of disjunction property within the calculation of the partition lattice. Indeed, if $\{z_1, z_2\}$ and $\{z_3\}$ overlap, every partition verifying the pattern $\{\{z_1, z_2\}, \{z_3\}, X\}$, with X a partition of the complement to $\{z_1, z_2, z_3\}$ are trivially not disjoint. This observation allows to prune large parts of the *Hasse diagram* of the set of candidate summary partitions $\mathbf{P}$ and could drastically reduce time cost.

Besides, note that the uniqueness of such optimal partitioning is not guaranteed. To address this problem, we can select in both *GO-GMA* and *LO-GMA* the overall hierarchy that has the largest area under the U curve (see Figure 2.5, Chapter 2). It is given by the sum of U values in every level of the tree.

Management of massive data sets requires efficient algorithms to merge summary hierarchies. The *GMA* approaches provide, by construction, optimal hierarchies but they suffer from an exponential complexity and are thereafter inappropriate for scaling.

3.1.3.3 SEQ-based Merging Algorithm (*SEQ-MA*)

One direction to overcome the *GMA* limitation (i.e., exponential computational complexity) is to process the most specialized summary partitioning of distributed data using the SAINTE-

[27] $\mathcal{B}_L = \sum_{i=0}^{L-1} (C_i^{L-1} \cdot \mathcal{B}_i)$ gives the number of partitions of a set $\mathcal{Z}$ with L elements according to the usual definition.

TIQ summarization service (Section 2.1.1.2, Chapter 2). We will refer to this approach as the *SEQ-based Merging Algorithm* (*SEQ-MA*) for the remainder of this chapter.

According to the example described in section 3.1.2, the result hierarchy of the summary hierarchies H_{R_1} and H_{R_2} according to the *SEQ-MA* approach is the same as the one provided by SAINTETIQ (i.e., mapping and summarization services) when performed on the global data set R (Figure 3.2). Note that this is not always the case since different sorts of cells may yield different hierarchies [SPRM05].

SEQ-MA relies on the SAINTETIQ summarization service and hence its time complexity T_{SEQ-MA} is $O(L \cdot \log L)$, where L is the number of cells populated by records of the global data set.

Note that *SEQ-MA* considers the global data set in an attempt to build the global summary hierarchy. Indeed, it does not exploit existing hierarchical partitioning schemas that are pre-computed locally ; it takes a set of cells as input and builds the global summary hierarchy from scratch. Thus, *SEQ-MA* is expected to break down rapidly as the data size and dimensionality increase (see experimental results in Section 3.4).

In contrast, the following alternatives try to achieve high quality clustering schemas of the entire distributed data set as well as to enhance merging process performance, exploiting the capacity of distributed resources.

3.2 Alternative approaches

3.2.1 Horizontally Distributed Data

In this section, we present two alternative algorithms, which deal with horizontally distributed data, in order to overcome the limitations of *GMA* and *SEQ-MA* approaches. The first proposal, the *Union by Incorporation Algorithm* (*UIA*), modifies one of the two input trees, called the base model, according to the other one to build a single tree. The second approach, so-called *Union by Alignment Algorithm* (UAA), consists in rearranging summaries by levels in a top-down manner.

3.2.1.1 Union by Incorporation Algorithm (*UIA*)

Howto

This method is based on the incremental conceptual clustering algorithm (summarization service) used by SAINTETIQ : leaves (or cells) of a summary tree are introduced into the other

hierarchy one at a time with a top-down approach and they are incorporated into best fitting nodes descending the tree. Indeed, at each node z, the algorithm considers *incorporating* the current cell c into each child node of z, updating the description of the targeted summary. Furthermore, the process evaluates the preference for *merging* two children nodes of z, *splitting* one child node and *creating* a new child node accommodating c. Then it uses the U score to rank candidate partitions for $z \widetilde{\cup} c$ and it changes the state of the hierarchy according to the best operator to apply each time a new cell descending the tree encounters a summary. Once the new cell c reaches the leaves, if a duplicate exists, then it is updated regarding information of the new cell, else, we develop the tree with a new level such that leaves remain single cells only. The previous leaf becomes an intermediate node with two cells, the new leaves, as children nodes.

According to the example described in Section 3.1.2, Figure 3.9 represents the hierarchy produced when H_{R_2} is incorporated into H_{R_1}.

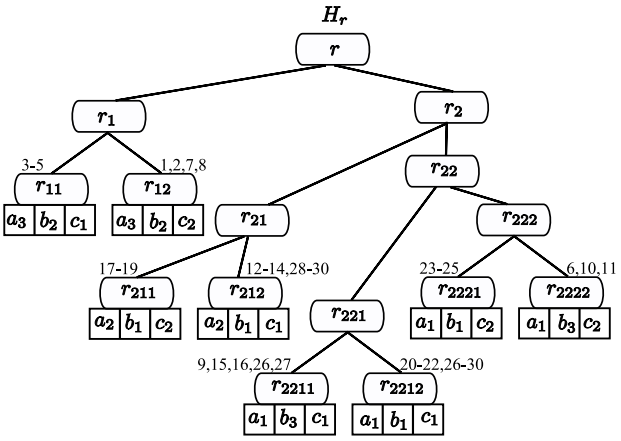


Figure 3.9 – *UIA*$_{2in1}$ (i.e., H_{R_2} in H_{R_1})

Discussion

Since the *UIA* approach is based on the SAINTETIQ summarization service, its time complexity T_{UIA} is also $O(L \cdot \log L)$, where L is the number of cells populated by records of the global data set. Note that T_{UIA} is in the same order of magnitude than T_{SEQ-MA} (Section 3.1.3.3). However, performance enhancement is truly remarkable (see experimental results in Section 3.4) since only leaves of one input hierarchy are processed.

Besides, *UIA* is natively asymmetrical. Indeed, it assumes one of the two input trees as the base model and the other one is deconstructed to incorporate every single leaf into the first tree. Thus, an important issue for performance and quality requirements of the result hierarchy is

to define the base tree for the operation. The basic assumption we made is that we prefer the largest tree to be the base model since it is expected to be stable or at least not so far from the stable state. Hence, in the best case, *UIA* performs a classification task only.

In the following section, we propose a second approach, called *Union by Alignment Algorithm* (*UAA*), for summarizing horizontally distributed data. *UAA* is drastically different than the *UIA* approach. Indeed, *UAA* is inspired of tree alignment techniques whereas *UIA* is based on the SAINTETIQ engine. Furthermore, *UAA* fully reuses the existing hierarchies, whereas *UIA* reuses only the base model. Thus, *UAA* is appropriate for situations where both hierarchies are very large.

3.2.1.2 Union by Alignment Algorithm (*UAA*)

Howto

The *Union by Alignment Algorithm* (*UAA*) is based on a recursive algorithm which operates unions of summaries guided by nodes of the input hierarchies H_{R_1} and H_{R_2} in order to produce the global hierarchy $H_R = H_{R_1 \cup R_2}$.

The result hierarchy H_R (rooted by $z^\star$) is built from trees H_{R_1} (rooted by z_1) and H_{R_2} (rooted by z_2) as follows :

1. $z^\star \leftarrow z_1 \widetilde{\cup} z_2$;
2. compute P_{best} of $P = P_{z_1} \widehat{\cup} P_{z_2}$;
3. connect P_{best} to $z^\star$ as the top-level of H_R ;
4. for each $z \in P_{best}$ do
   - place in H_R the sub-tree rooted by z if z is a single summary in H_{R_1} or H_{R_2} ; then stop ; else (i.e., there exists a set of summaries F that are single summaries in H_{R_1} or H_{R_2} such that $z = \widetilde{\cup}_{z' \in F} z'$)
   - apply recursively *UAA* from step two with $P \leftarrow \widehat{\cup}_{z' \in F} P_{z'}$; $z^\star \leftarrow z$; $H_R \leftarrow H_{R_z}$;

Recall that P_z denotes the set of children nodes of z, and it is a summary partitioning of R_z, the extent of z.

Figure 3.10 represents the *UAA* hierarchy of the input hierarchies shown in Figure 3.3.

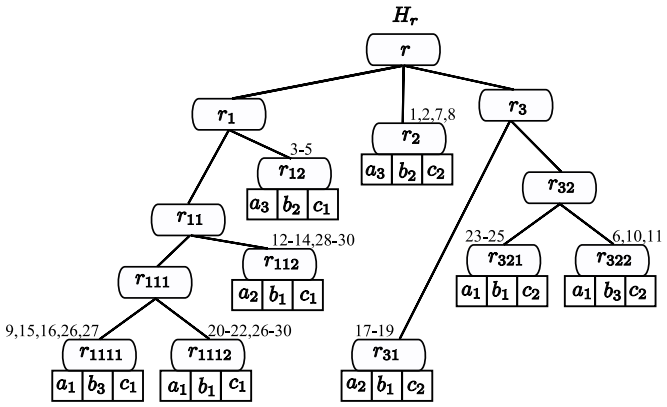


Figure 3.10 – *UAA* of H_{R_1} and H_{R_2}

Discussion

The time complexity T_{UAA} of *UAA* is linear w.r.t. the number of cells L populated by records of the global data set. It is given by :

$$T_{UAA}(L) = k_{UAA} \cdot \mathcal{B}_d \cdot \frac{L-1}{d-1} \in O(L)$$

where k_{UAA} is a constant factor and d is the average degree of the output tree. In the above formula, $\frac{L-1}{d-1}$ corresponds to the number of the output tree nodes, whereas $\mathcal{B}_d$ (the dth *Bell number*) gives an estimation of the time required to compute P_{best} of $P = P_{z_1} \widehat{\cup} P_{z_2}$.

Note that in the second step of the algorithm, the best partitioning P_{best} of the set $P = P_{z_1} \widehat{\cup} P_{z_2}$ of summaries that specializes $z^\star$ can be a singleton (i.e., $P_{best} = \{z^\star\}$). For instance, it occurs when all summaries of $P_{z_1} \cup P_{z_2}$ are pairwise overlapping. In that case, we look for the best partitioning in a specialization of P instead of P itself, that is we replace part of the summaries of P by their children and we continue recursively until we find a non trivial (cardinality greater than or equal to 2) partitioning. In the worst case, the specialization stops when P contains all leaves of hierarchies H_{R_1} and H_{R_2}. The best partitioning of P is then the set of leaves with duplicate elimination since at this stage it is the unique partition of $z^\star$ that satisfies disjunction property.

Currently, when this happens we simply try to specialize one summary that is not a leaf. However, we can use any other heuristic function to support specialization. For example, we could evaluate U on a given subset of specialized partitions and retain the highest U-valuated candidate only.

3.2.2 Vertically Distributed Data

Both *UIA* and *UAA* are not suitable for vertically fragmented data. Indeed, they rely on the summary union operator $\widetilde{\cup}$ that aggregates two input summaries representing two different sets of database records with the same set of attributes. Since summaries are represented in different feature spaces, $\widetilde{\cup}$ is no more applicable for joining them. Moreover, similarity measures between summaries could not even be defined over two different feature spaces. Furthermore, the join of two summaries can be empty, whereas the union of two summaries cannot. In this section, we present three new algorithms to manage vertically distributed data. The *Subspace-Oriented Join Algorithm* (*SOJA*) and the *SOJA with Rearrangements Algorithm* (*SOJA-RA*) modify one of the two input trees, called the base model, according to the other one to build a single tree. The last approach, called *Tree Alignement-based Join Algorithm* (*TAJA*), relies on a recursive processing that performs a join of summary partitions guided by levels of the input hierarchies.

3.2.2.1 Subspace-Oriented Join Algorithm (*SOJA*)

Howto

The main idea of this approach is rather simple. It starts from an existing summary hierarchy within a subspace of the whole data set. Then, once items become indistinguishable from one-another according to this subspace, it proceeds with a sequence of refinements on the existing clusters according to the complementary subspace. More precisely, *SOJA* assumes one of the two input trees as the base model (e.g., $H_{R'_1}$) and the other one (e.g., $H_{R'_2}$) is processed to refine cells (or leaves) of the first one.

Thus, $SOJA_{1\rightarrow 2}$ (i.e., $H_{R'_1}$ is the base model) computes the hierarchy $H_R = H_{R'_1 \bowtie R'_2}$ from $H_{R'_1}$ and $H_{R'_2}$ as follows :

1. $H_R \leftarrow H_{R'_1}$;
2. for each cell c_1 of H_R, compute $L_{R'_2} \widehat{\ltimes} \{c_1\}$ and do
   a. if $L_{R'_2} \widehat{\ltimes} \{c_1\} = \emptyset$: remove c_1 from H_R since it is populated by records that are not in $R'_1 \bowtie R'_2$ and then, if the parent of c_1 has one single child, replace it by the child itself, else
   b. if $L_{R'_2} \widehat{\ltimes} \{c_1\} = \{c_2\} : c_1 \leftarrow c_1 \widetilde{\bowtie} c_2$, else
   c. process the set of cells $L_{R'_2} \widehat{\ltimes} \{c_1\}$ using the SAINTETIQ summarization service and replace each node z of the resulting hierarchy by $c_1 \widetilde{\bowtie} z$ then, replace c_1 with the new resulting hierarchy ;

3. for each node z of H_R, build z's intent on the overall features set based on z's cells, i.e., $z = \sum_{c \in L_z} c$.

The operator $\sum$ stands for the union of the descriptions of cells to build the including hyperrectangle as a resulting summary.

The computation of $L_{R'_2} \widehat{\ltimes} \{c_1\}$ is based on a depth-first search and relies on a strong property of the hierarchy : the generalization step in the SAINTETIQ model guarantees that a tuple is absent from a summary's extent if and only if it is absent from any partition of this summary. This property of the hierarchy permits branch cutting as soon as it is known that no result will be found. Regarding the processed cell (of the base tree), only a part of the hierarchy is explored.

Figure 3.11 illustrates the joined hierarchy obtained from summary trees shown in Figure 3.4 according to $SOJA_{1\to2}$.

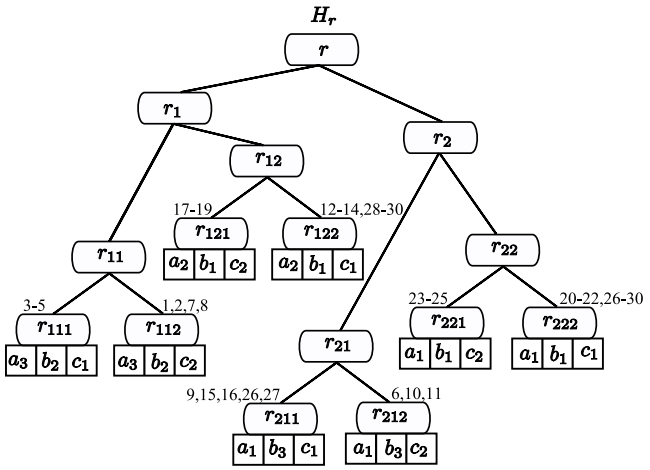


Figure 3.11 – SOJA$_{1\to2}$ on $H_{R'_1}$ and $H_{R'_2}$

Discussion

Assume that $l = |L_{R'_1}|$ and $|L_{R'_1}| \simeq |L_{R'_2}|$, where $|.|$ denotes the cardinality of a set. The time complexity T_{SOJA} of this approach is defined as follows :

$$T_{SOJA}(l) = l \cdot [k_{SOJA} \cdot \frac{l-1}{d-1} + T_{SEQ}(l)] + k'_{SOJA} \cdot \frac{l^2-1}{d-1}$$

where k_{SOJA} and k'_{SOJA} are constant factors and d is the average width of $H_{R'_1}$ and $H_{R'_2}$. In the above formula, $[k_{SOJA} \cdot \frac{l-1}{d-1} + T_{SEQ}(l)]$ is the time cost required to process each cell of the base tree (i.e., *search* for cells from the second tree that would join with the current cell and *cluster*

them using SAINTETIQ summarization service), l^2 gives an upper bound of the number of cells populated by records of $R'_1 \bowtie R'_2$ and $[k'_{SOJA} \cdot \frac{l^2-1}{d-1}]$ gives an estimation of the time required to update the description of the global hierarchy.

As one can see, the computational complexity of *SOJA* is in the same order of magnitude than that of *SEQ-MA* (Section 3.1.3.3), i.e., $O(L \cdot \log L)$ where L is the number of cells populated by records of $R'_1 \bowtie R'_2$. However, the performance gain is very noticeable (see experimental results in Section 3.4) since all required clustering schemas are computed within a low-dimensional feature space (i.e., E_2 in the case of $SOJA_{1\to 2}$).

In the next section, we propose a modified version of *SOJA*, called *SOJA with Rearrangements* (*SOJA-RA*), that aims to fully exploit pre-computed hierarchies. In fact, *SOJA-RA* uses the same algorithm described in Section 2.4 of Chapter 2 to provide the hierarchical clustering schema of $L_{R'_2} \widehat{\ltimes} \{c_1\}$.

3.2.2.2 SOJA with Rearrangements (*SOJA-RA*)

Howto

At step $2.c.$ of the *SOJA* algorithm, we can also use the existing hierarchy $H_{R'_2}$ to provide a hierarchical clustering schema of $L_{R'_2} \widehat{\ltimes} \{c_1\}$. Indeed, starting from the set of cells $L_{R'_2} \widehat{\ltimes} \{c_1\}$, we can produce a sequence of nested partitions with a decreasing number of clusters. Each partition results from the previous one by merging the 'closest' clusters into a single one. Similar clusters are identified thanks to the hierarchical structure of the pre-computed summary hierarchy $H_{R'_2}$. The general assumption is that summaries which are closely related have a common ancestor lower in the hierarchy, whereas the common ancestor of unrelated summaries is near the root. This process stops when it reaches a single cluster (the root $z^\star$). It is worth noticing that $z^\star$ is built at the same time we search for cells from $H_{R'_2}$ that would join with c_1. It means that no clustering at all would have to be performed : we prune the tree $H_{R'_2}$ by retaining only leaves that belong to $L_{R'_2} \widehat{\ltimes} \{c_1\}$ and inner nodes that have two or more cells from $L_{R'_2} \widehat{\ltimes} \{c_1\}$ as descendant nodes.

According to the example described in Section 3.1.2, the result hierarchy of the summary hierarchies $H_{R'_1}$ and $H_{R'_2}$ according to $SOJA\text{-}RA_{1\to 2}$ is the same as the one provided by $SOJA_{1\to 2}$ (Figure 3.11). We note though that this is not always the case.

Discussion

Assume that $l = |L_{R'_1}|$ and $|L_{R'_1}| \simeq |L_{R'_2}|$. The computational complexity $T_{SOJA-RA}$ of *SOJA* when using the above process (*Rearranging Summaries Algorithm*) is given by :

$$T_{SOJA-RA}(l) = k_{SOJA-RA} \cdot l \cdot \frac{l-1}{d-1} + k'_{SOJA} \cdot \frac{l^2-1}{d-1}$$

where $k_{SOJA-RA}$ and k'_{SOJA} are constant factors and d is the average width of $H_{R'_1}$ and $H_{R'_2}$. Notice that this formula is identical to that of T_{SOJA}, except that the first term is replaced by $[k_{SOJA-RA} \cdot l \cdot \frac{l-1}{d-1}]$, the time cost required to build the clustering schemas corresponding to the cells of the base tree using the above rearranging process.

SOJA-RA is more efficient than *SOJA*. Indeed, its computational cost is $O(L)$, whereas for *SOJA* the cost is $O(L \cdot \log L)$, where L is the number of cells populated by records of $R'_1 \bowtie R'_2$. This is due to the fact that *SOJA-RA* fully reuses the existing hierarchies, whereas *SOJA* reuses only the base model.

Note that *SOJA* and *SOJA-RA* are asymmetrical. Indeed, they assume one of the two input trees as the base model and the other one is processed to refine cells of the first one. This implies that clusters are discovered first based on the feature set of the base model (the upper part of the tree) and then based on the complementary feature set (the lower part). Thus, we obtain two different hierarchies according to the base model ($H_{R'_1}$ or $H_{R'_2}$). None of them is preferred to the other and the third-party application or user's requirements are key factors for selecting the appropriate base model. Consider a bank's database with two relations : a relation Customers (R'_1) with attributes `Id_Customer`, `Age` and `Income` and, a relation Banking_Products (R'_2) with attribute `Id_Customer`, `Number_of_Accounts` and `Number_of_Credit_Cards`. For instance, $SOJA_{2\rightarrow 1}$ or $SOJA\text{-}RA_{2\rightarrow 1}$ (i.e., $H_{R'_2}$ is the base model) are more relevant choices given the following banker's request "how customers with many credit cards and only one account are clustered according to their age and income ?".

However, defining clusters in terms of simultaneous closeness on all features may sometimes be desirable. In such cases, the *Tree Alignement-based Join Algorithm* (*TAJA*), described in the following, is more relevant. Indeed, *TAJA* consists in rearranging summaries by levels in a top-down manner and consequently, features have an equal influence in the clustering process.

3.2.2.3 Tree Alignement-based Join Algorithm (*TAJA*)

Howto

The *Tree Alignement-based Join Algorithm* (*TAJA*) consists in a recursive processing that performs joins of summary partitions guided by levels of the input hierarchies $H_{R'_1}$ and $H_{R'_2}$ in order to produce the global hierarchy $H_R = H_{R'_1 \bowtie R'_2}$.

The result hierarchy H_R (rooted by $z^\star$) is built from trees $H_{R'_1}$ (rooted by z_1) and $H_{R'_2}$ (rooted by z_2) as follows :

1. $z^\star \leftarrow z_1 \widetilde{\bowtie} z_2$;
2. compute $P = P_{z_1} \widehat{\bowtie} P_{z_2}$; P is then the top-level partition in the tree H_R and is connected to the root $z^\star$;
3. for each $z \in P$ (i.e., $\exists\ z'_1 \in P_{z_1}$ and $\exists\ z'_2 \in P_{z_2}$ such that $z = z'_1 \widetilde{\bowtie} z'_2$) do :
   - nothing if z is a leaf, else
   - apply recursively *TAJA* from step one with $z_1 \leftarrow z'_1$; $z_2 \leftarrow z'_2$; $H_R \leftarrow H_{R_z}$;
4. for each node z of H_R, build z's intent on the overall features set based on z's cells, i.e., $z = \sum_{c \in L_z} c$.

Figure 3.12 represents the *TAJA* hierarchy of the input hierarchies shown on Figure 3.4.

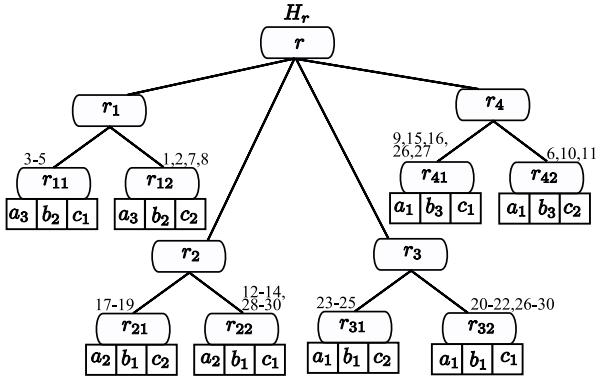


Figure 3.12 – TAJA on $H_{R'_1}$ and $H_{R'_2}$

Discussion

The time complexity T_{TAJA} of *TAJA* is linear w.r.t. the number of cells L populated by records of the global data set.

T_{TAJA} is given by :

$$T_{TAJA}(L) = k_{TAJA} \cdot \frac{L-1}{d-1} \cdot d^2 + k'_{TAJA} \cdot \frac{L-1}{d-1} \in O(L)$$

where k_{TAJA} and k'_{TAJA} are constant factors and d is the average degree of the output tree. In the above formula, d^2 gives an estimation of the time required to compute $P = P_{z_1} \widehat{\bowtie} P_{z_2}$, $[k_{TAJA} \cdot \frac{L-1}{d-1} \cdot d^2]$ is the time cost required to build hierarchical structure of the global summary and $[k'_{TAJA} \cdot \frac{L-1}{d-1}]$ is the time required to update its description.

3.2.3 Highlights of the Proposed Algorithms

To sum up so far, we have proposed five algorithms (*UIA*, *UAA*, *SOJA*, *SOJA-RA* and *TAJA*) for merging summary hierarchies. They can be categorized as follows :

- ***Horizontal vs. Vertical partitioning of data :*** *UIA* and *UAA* deal with horizontally distributed data, whereas *SOJA*, *SOJA-RA* and *TAJA* manage vertically distributed data ;
- **SAINTETIQ*-based vs. Tree-based merging approaches :*** *UIA* and *SOJA* rely on the SAINTETIQ engine, whereas *UAA*, *SOJA-RA* and *TAJA* rely on tree merging techniques ;
- ***Complete vs. Partial reuse of local models :*** *UAA*, *SOJA-RA* and *TAJA* fully reuse the input hierarchies, whereas *UIA* and *SOJA* reuse one of the input hierarchies (i.e., the base model) ;
- ***Asymmetric vs. Symmetric approaches :*** *UIA*, *SOJA* and *SOJA-RA* assume one of the two input hierarchies as the base model, whereas *UAA* and *TAJA* make no distinction between the input hierarchies ;
- ***Linear vs. Quasi-Linear complexity :*** *UAA*, *SOJA-RA* and *TAJA* have linear complexity $O(L)$, whereas *UIA* and *SOJA* have quasi-linear complexity $O(L \cdot \log L)$, where L is the number of cells populated by records of the global data set.

Recall that all these algorithms take as input two distinct summary hierarchies (local models) and generate a single summary hierarchy of the entire distributed data. Thus, if we have more than two input hierarchies defined over the same feature set (resp., different sets of features), the global hierarchy can be computed by simple repetition of the binary union (resp., join) process, i.e., we compute repeatedly the union (resp., join) of any two of them to get an intermediate hierarchy, and so on.

Finally, given that our merged hierarchies and local hierarchies share the same structure (i.e., a grid-based clustering schema), any grid-based clustering method that enables the management of local updates, can be used to reflect these updates on the global schema. For instance,

SEQ-MA, which relies on the SAINTETIQ model, applies a conceptual clustering algorithm for partitioning the global cells (the most specialized summaries) in an incremental and dynamic way. Thus, changes in local models could be reflected through such an incremental maintenance of the global hierarchy [SPRM05].

In the following, we discuss an important issue for merging processes regarding the quality assessment of the results.

3.3 Joining validity assessment

In this work, we aim at merging two hierarchical clustering schemas. The question we wish to answer in this section is — *how good are our merged hierarchies ?*

3.3.1 Background

In [HBV01], a number of clustering techniques and algorithms have been reviewed. These algorithms behave in different ways depending on the features of the data set (geometry and density distribution of clusters) and/or the input parameter values (e.g., number of clusters, diameter or radius of each cluster). Thus, the quality of clustering results depends on the setting of these parameters.

The soundness of clustering schemas is checked using validity measures (indices) available in the literature [HBV01]. Indices are classified into three categories : external, internal, and relative. The first two rely on statistical measurements and aim at evaluating the extent to which a clustering schema maps a pre-specified structure known about the data set. The third category of indices aims at finding the best clustering schema that an algorithm can provide under some given assumptions and parameters. As one can observe, these indices give information about the validity of the clustering parameters for a given data set and thus may be viewed as data dependent measures.

Recall that we try to evaluate the validity of the merging algorithms (*UIA*, *UAA*, *SOJA*, *SOJA-RA* and *TAJA*). Therefore usual validity measures do not apply since the main purpose of the evaluation is not to evaluate the clustering algorithm of the SAINTETIQ system, but rather to compare the distributed summary construction process with the centralized approach, everything else being equal (objective function, parameters, grid and data set).

Within this framework, we consider hierarchies $H_{GO\text{-}GMA}$ and $H_{LO\text{-}GMA}$ provided by the *GMA* approach as the reference hierarchies. Then, we compare the hierarchies provided by *UIA*, *UAA*,

SOJA, *SOJA-RA*, *TAJA* and the basic centralized approach of SAINTETIQ with those reference hierarchies. Thus, there is a need for a quality measure of summary tree.

A valid and useful hierarchy quality measure must be :

- data independent, that is, not built according to pre-specified data structure, assumptions and parameters ;
- maximum for the reference hierarchies provided by the *Greedy Merging Algorithm*.

In the following, we define two different measures according to the reference hierarchy considered ($H_{GO\text{-}GMA}$ or $H_{LO\text{-}GMA}$). The first one relies on a level-based analysis and is relevant when the reference hierarchy is produced with the global optimization. The second measure is well-suited for locally optimized hierarchies.

3.3.2 Level-based Analysis

The basic idea is to study the summary utility (U) level by level as the *GO-GMA* does. For a given hierarchy H, we then define as many partitions as there are levels, from the root (P_0) to the leaves (P_h), where h is the height of the tree. P_{i+1} is the direct specialization of every summary in P_i, except for the leaves. And every $P_i, i \in [0, h]$ satisfies both the disjunction and coverage properties. Thus, we associate to each level i of the tree the utility value $\mathsf{U}(P_i)$ of the related partition P_i and consequently an utility vector $\langle \mathsf{U}(P_0), \ldots, \mathsf{U}(P_h) \rangle$.

Then, the overall score ξ of H is defined as the sum of all those utilities (the "area under the curve" of summary utility) :

$$\xi(H) = \sum_{i=0}^{h} \mathsf{U}(P_i)$$

Table 3.1 – Utility vectors and ξ-values

$H \backslash l$	0	1	2	3	4	$\xi(H)$
$GO\text{-}GMA$	0.83	0.87	0.63	0.41	0.27	3.01
UIA_{2in1}	0.83	0.87	0.47	0.39	0.27	2.83
SEQ	0.83	0.85	0.51	0.27	0.27	2.73
UAA	0.83	0.80	0.49	0.33	0.27	2.72
$SOJA_{1\rightarrow 2}$	0.83	0.85	0.50	0.27	0.27	2.72
$LO\text{-}GMA$	0.83	0.87	0.27	0.27	0.27	2.51
$TAJA$	0.83	0.50	0.27	0.27	0.27	2.14

Table 3.1 gives utility vectors and ξ-values according to each approach when applied to summary trees introduced in Section 3.1.2. Levels range from zero to four, and *SEQ* stands for the SAINTETIQ process.

UIA, *UAA*, *SOJA*, *SOJA-RA* and *TAJA* algorithms give values comparable to the one provided by SAINTETIQ. Thus, even if theses algorithms have not been designed to be globally optimal, they offer a similar quality regarding the ξ measurement. It means that merging processes are, from the GO point of view, as effective as the centralized version of summary construction.

Furthermore, it is worth noticing that, as expected, this measure is not adapted to local analysis. Indeed, it does not give a maximum for the hierarchy $H_{LO\text{-}GMA}$. To address this problem, we then present in the following section a new dissimilarity measure between summary trees.

3.3.3 Summary Tree Dissimilarity

The basic idea is to valuate local differences between two hierarchical clustering schemas. Assume a reference summary tree H_{ref} ; each node z covers a part R_z of relation R, and provides a partitioning P_z of R_z thanks to the top-level of the sub-tree rooted by z in H_{ref}. Given a targeted tree H_{target}, we then propose to valuate the partitioning of R_z, for each z in H_{ref}, according to H_{target}. In other words, we try to locate R_z inside H_{target} and compose the partitioning P'_z of R_z w.r.t. summaries from H_{target}. Hence, we compute the utility value for P'_z to compare with the utility value of P_z. Among all the candidate partitioning of R_z in H_{target}, we retain the most generalized one thanks to a bottom-up traversal algorithm.

The above process provides two utility vectors : $v = \langle q_1, \ldots, q_n \rangle$ with values related to H_{ref} and $v' = \langle q'_1, \ldots, q'_n \rangle$ for H_{target}. Notice that the dimension of these vectors is equal to the number n of intermediate nodes in the reference hierarchy H_{ref}.

Thus, a dissimilarity measure between H_{ref} and H_{target} is defined as the distance between the above utility vectors v and v'. As an example, the Euclidian distance is used here :

$$\delta(H_{target}, H_{ref}) = \sqrt{\sum_{i=1}^{n} (q'_i - q_i)^2}$$

Even if it relies on a distance, the dissimilarity measure δ is asymmetric since the utility vector v' of H_{target} is relative to data subsets provided by H_{ref}.

In the following, we use the hierarchy $H_{LO\text{-}GMA}$, provided by the *GMA* process based on local optimization, as the reference hierarchy ($H_{ref} = H_{LO\text{-}GMA}$). Denote by $\delta_{LO}(H) = \delta(H, H_{LO\text{-}GMA})$ the dissimilarity measure between a hierarchy H and the locally optimized one

$H_{LO\text{-}GMA}$. Note that by construction, for each summary z of $H_{LO\text{-}GMA}$ and for any hierarchy H_{target} we have $P_z = P_{best}$ and consequently $\mathsf{U}(P_{worst}) \leq \mathsf{U}(P'_z) \leq \mathsf{U}(P_z)$.

Hence, as shown in Table 3.2, the dissimilarity measure is semantically consistent. H_{worst} stands for the hierarchy provided by *LO-GMA* with a choice of the worst (instead of the best) partition at each step of the process. According to the example described in Section 3.1.2, H_{worst} has only two levels : one root level and one cells level.

Table 3.2 – δ_{LO}-values

H	$H_{GMA_{LO}}$	H_{SEQ}	$H_{UIA_{2in1}}$	$H_{GMA_{GO}}$	$H_{SOJA_{1\to 2}}$	H_{TAJA}	H_{UAA}	H_{random}	H_{worst}	$H_{\emptyset}$
$\delta_{LO}(H)$	0	0.45	0.45	0.46	0.48	0.77	1.02	1.35	1.41	1.56

Moreover, if we execute *GMA* with local optimization 100 times with random choice of partition at each step of the process, the average of all δ_{LO} outputs is 1.35 that is very greater than 1.02, the worst δ_{LO} of all merged hierarchies. It allows us to conclude that *UIA*, *UAA*, *SOJA*, *SOJA-RA* and *TAJA* provide well-founded summary trees.

3.3.4 Discussion

The SAINTETIQ system is designed for converging to the locally optimized hierarchy since the tree is updated at each node every time a new cell is incorporated [SPRM05]. Thus, the summary tree dissimilarity measure δ_{LO} is more adapted than the ξ measure to evaluate the validity of our merging algorithms.

However, δ_{LO} relies on the hierarchy provided by the *LO-GMA* approach that is very difficult (even impossible) to compute for a large dataset. Furthermore, notice that it would not be appropriate to use another hierarchy (e.g., the SAINTETIQ hierarchy) as a reference one since the dissimilarity measure δ is asymmetric, and therefore δ's results cannot be explained and interpreted. In fact, the merged hierarchies may be inferior, equal, or better in quality than the hierarchy provided by SAINTETIQ when performed on the global dataset.

To address this problem, we propose to study the summary utility per node (i.e., locally) as the *LO-GMA* does. For a given hierarchy H_R, we then define as many partitions as there are non-leaf nodes ; each node z covers a part R_z of relation R, and provides a partitioning P_z of

R_z. Thus, we associate to each node z the utility value $\mathsf{U}(P_z)$. Consequently, we obtain as many utility values as there are non-leaf nodes in H_R.

Then, we define the summary tree quality σ_k of a tree H_R as follows :

$$\sigma_k(H_R) = \frac{\sum_{z \in k\text{-nodes}} \mathsf{U}(P_z)}{|k\text{-nodes}|}$$

where k-nodes is the set of nodes in H_R with depth less or equal than k. Note that $\sigma_0(H_R)$ is the utility value of the top-level partition in H_R since 0-nodes $= \{root\}$.

The above measure allows us to evaluate how well the local optimization objective is fulfilled. Table 3.3 gives σ_k values according to each approach, with $k \in \{1, 2\}$.

Table 3.3 – σ-values

$k \backslash H$	$H_{GMA_{LO}}$	H_{SEQ}	$H_{UIA_{2in1}}$	$H_{GMA_{GO}}$	$H_{SOJA_{1\rightarrow 2}}$	H_{TAJA}	H_{UAA}	H_{worst}
1	0.76	0.58	0.50	0.46	0.47	0.39	0.41	0.27
2	0.76	0.49	0.40	0.37	0.36	ND	0.35	0.27

As one can observe, the summary tree quality σ_k also reaches its maximum and minimum values on $H_{GMA_{LO}}$ and H_{worst} respectively. We will therefore use it in the following section to evaluate the validity of our merging algorithms when applied to a more extensive data set.

3.4 Experimental Results

This section presents experimental results achieved with the *UIA*, *UAA*, *SOJA*, *SOJA-RA* and *TAJA* algorithms. We first introduce the data set, then we provide an analysis based on observations of various parameters.

3.4.1 Data Set

We used a data set generator (DatGen[28]) to generate synthetic[29] data sets with different numbers of records. Each record is defined over $N = 20$ attributes (i.e., $E = \{A_1, \ldots, A_{20}\}$)

[28] www.datasetgenerator.com

[29] The use of synthetic data, where we can set several parameters (e.g., the number of attributes, the number of cells, the fragmentation rate), instead of real data sets allows us to more exhaustively evaluate how our merging algorithms would behave under different conditions.

with values from a set of 10 nominal values that simply serve as linguistic labels, and it has a primary key ID.

To perform our merging processes, we previously computed (using SAINTETIQ without any mapping) four sets of couples of hierarchies :

- the first set contains couples (I_1, I_2) such that R_{I_1} maps 50 cells and R_{I_2} maps 100, 200, 400, ... and 2000 cells. R_{I_1} and R_{I_2} are defined over the same set of features $E = \{A_1, \ldots, A_{20}\}$;
- the second set contains couples (J_1, J_2) such that $D = R_{J_1} \bowtie_{\mathsf{ID}} R_{J_2}$ maps $L \in \{200, 400, \ldots, 4000\}$ cells and J_1 summarizes D over the first 2 attributes (i.e., $E_1 = \{A_1, A_2\}$), whereas J_2 summarizes it over the remaining features (i.e., $E_2 = \{A_3, \ldots, A_{20}\}$). Thus, the number of cells of J_2 that would join with each cell of J_1 is high ;
- the third set contains couples (H_1, H_2) such that $D' = R_{H_1} \bowtie_{\mathsf{ID}} R_{H_2}$ maps $L = 10000$ cells and H_1 summarizes D' over the first N_1 attributes (i.e., $E_1 = \{A_1, \ldots, A_{N_1}\}$), whereas H_2 summarizes D' over the last $N - N_1$ attributes (i.e., $E_2 = \{A_{N_1+1}, \ldots, A_{20}\}$), where N_1 ranges from 1 to 19 ;
- The fourth set contains couples (K_1, K_2) such $D'' = R_{K_1} \bowtie_{\mathsf{ID}} R_{K_2}$ maps $L = 10000$ cells and K_1 summarizes D'' over $E_1 = \{X, Y\}$, whereas H_2 summarizes D'' over $E_2 = \{Y, Z\}$, where $\{X, Y, Z\}$ is a partition of $E = \{A_1, \ldots, A_{20}\}$ such that $|X| = |Z|$ and $|Y| \in \{4, 8, 12\}$.

All experiments were done on a 1.7GHz P4-based computer with 768MB memory.

3.4.2 Results

In this section, we validate our merging processes w.r.t. performance (computation times), structural properties (number of nodes and leaves, average depth and average width) and the summary tree quality measure σ_k.

3.4.2.1 Quantitative Analysis

From the analysis of theoretical complexities, we claim that *UAA* ($O(L)$), *UIA*$_{1in2}$ ($O(L \cdot \log L)$) and *UIA*$_{2in1}$ ($O(L \cdot \log L)$) are much faster than the the SAINTETIQ (*SEQ*) process ($O(L \cdot \log L)$) performed on $R_{I_1} \cup R_{I_2}$. That is the main result of Figure 3.13 that shows the performance evolution according to L, the number of cells populated by records of $R_{I_1} \cup R_{I_2}$. Observe that *UIA*$_{1in2}$ (i.e., I_2 is the base model) and *UAA* are much faster than *UIA*$_{2in1}$ (i.e., I_1 is the base model). The last one (*UIA*$_{2in1}$) is a bit more efficient than *SEQ*. This is due to the

fact that the base model I_1 is very small compared to I_2. Moreover, notice that the time cost of UIA_{1in2} is far less than others thanks to the stability of I_2 (the base model).

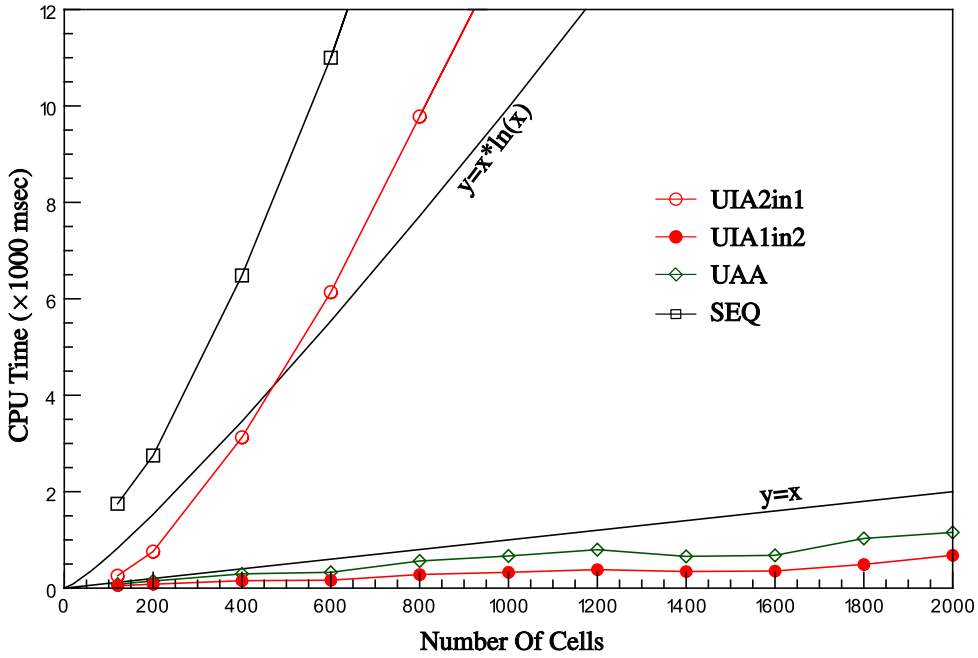


Figure 3.13 – Time cost comparison - (I_1, I_2)

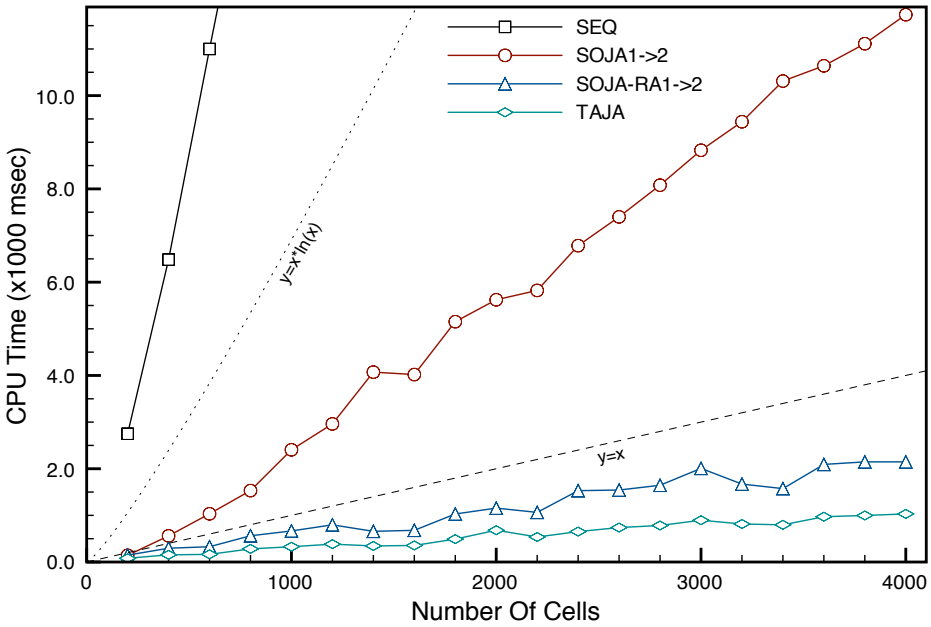


Figure 3.14 – Time cost comparison - (J_1, J_2)

Furthermore, Figure 3.14 shows the performance evolution of $\text{SOJA}_{1\to2}$, $\text{SOJA-RA}_{1\to2}$ and *TAJA*, when applied to (J_1, J_2), according to the number of cells populated by $R_{J_1} \bowtie R_{J_2}$.

As one can observe, these approaches are much faster than the *SEQ* process performed on $R_{J_1} \bowtie R_{J_2}$. Besides, *TAJA* and SOJA-RA$_{1\to2}$ are much more efficient than SOJA$_{1\to2}$ since (1) SOJA$_{1\to2}$ is based on the SAINTETIQ summarization service and (2) there are much correlations between R_{J_1} and R_{J_2} records, i.e., the number of cells that have to be clustered is very high. As one can observe, Figure 3.14 verifies experimentally that SOJA-RA$_{1\to2}$ and *TAJA* are linear ($O(L)$) w.r.t. the number of cells L of the global hierarchy whereas SOJA$_{1\to2}$ and *SEQ* are quasi-linear ($O(L \cdot \log L)$).

In the next experiment (see Figure 3.15), we use the third set of couples (i.e., (H_1, H_2)) to show how the CPU time of each joining process varies with changing the fragmentation rate ($\frac{N_1}{N} = \frac{|E_1|}{|E|}$). Observe that *TAJA* is a bit more efficient than SOJA-RA$_{1\to2}$. Further, as expected, the time cost of SOJA$_{1\to2}$ is quite similar to that of SOJA-RA$_{1\to2}$, except for $\frac{N_1}{N} \leq 0.20$. This is due to the fact that the number of cells of H_2 ($E_2 = \{A_{16}, \ldots, A_{20}\}$) that join with each cell of the base model H_1 ($E_1 = \{A_1, \ldots, A_4\}$) is high.

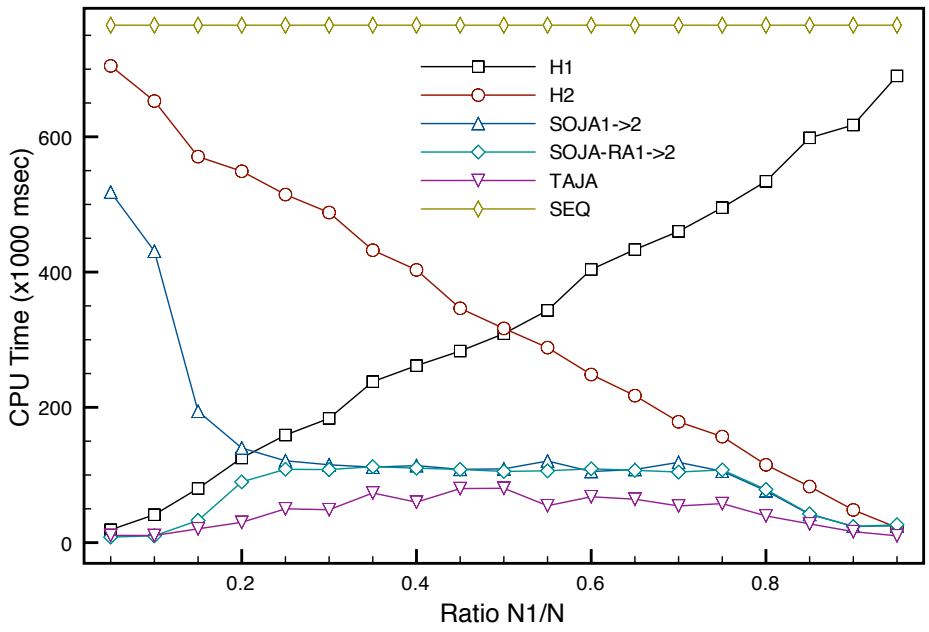


Figure 3.15 – Time cost comparison - (H_1, H_2)

Thus, the merging processes are able to drastically reduce the time cost of summarizing data sets of large size and dimensionality. This is achieved by horizontal and vertical fragmentation into several relations that would be summarized separately and then merged. Note however, that this is true only if we do not take into account the time cost of building local models.

3.4.2.2 Qualitative Analysis

In the following, the average depth and the average width of merged hierarchies are reported. For (I_1, I_2), we observe that :

- the average depth of the hierarchy provided by *UAA* and UIA_{1in2} (I_1 in I_2) is pretty much equal to the average depth of I_2 thanks to the stability of I_2. Moreover, UIA_{2in1} (I_2 in I_1) and the *SEQ* process provide hierarchies that are deeper than I_2. Average depth evolution of every hierarchy has been reported, according to the number of cells populated by records of the global data set $R_{I_1} \cup R_{I_2}$, in Figure 3.16 ;
- the hierarchies provided by *UAA* are wider than that generated with UIA_{1in2}. The last one is quite similar to I_2 and wider than the hierarchies provided by UIA_{2in1} and *SEQ* as well. Average width evolution of every hierarchy has been reported, according to the number of cells populated by records of $R_{I_1} \cup R_{I_2}$, in Figure 3.17 ;
- the number of nodes and leaves is quite similar through the merged and centralized hierarchies and is almost equal to the sum of I_1 and I_2 nodes and leaves.

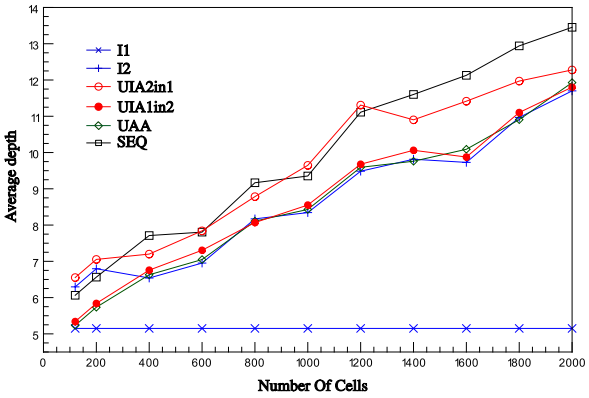


Figure 3.16 – Average depth comparison - (I_1, I_2)

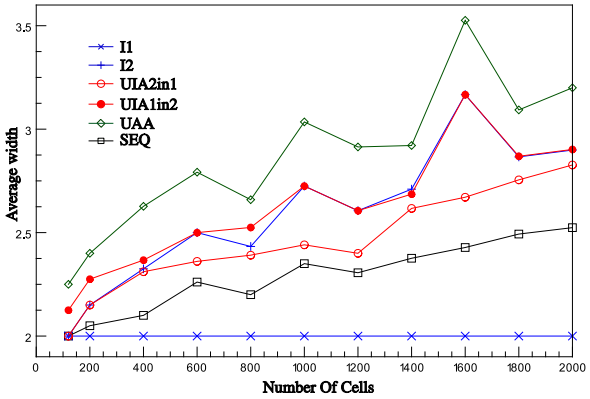


Figure 3.17 – Average width comparison - (I_1, I_2)

For (H_1, H_2), we observe that :

- the average depth of the hierarchy provided by $SOJA_{1\rightarrow 2}$ and $SOJA\text{-}RA_{1\rightarrow 2}$ (H_1 is the base model) is greater than that of the hierarchy provided by *SEQ*. The latter is also deeper than the one provided by *TAJA*. Average depth evolution of every hierarchy has been reported, according to the fragmentation rate ($\frac{N_1}{N}$), in Figure 3.18 ;
- the hierarchies provided by *TAJA* are wider than those generated with *SEQ*. However, the hierarchies provided by *SEQ* are wider than those provided by $SOJA_{1\rightarrow 2}$ and $SOJA\text{-}RA_{1\rightarrow 2}$. Average width evolution of every hierarchy has been reported, according to $\frac{N_1}{N}$, in Figure 3.19 ;

- the number of nodes is quite similar for SOJA$_{1\rightarrow 2}$, SOJA-RA$_{1\rightarrow 2}$ and *SEQ* hierarchies and is greater than that of the hierarchy provided by *TAJA*.

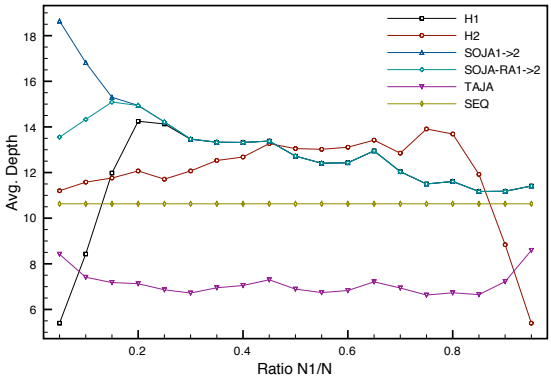


Figure 3.18 – Average depth comparison - (H_1, H_2)

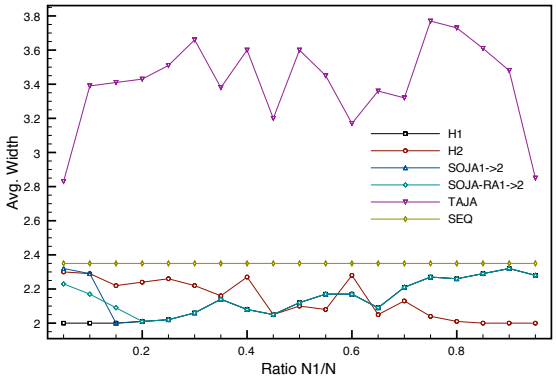


Figure 3.19 – Average width comparison - (H_1, H_2)

Note that for $\frac{N_1}{N} \geq 0.30$, H_1 and hierarchies provided by SOJA$_{1\rightarrow 2}$ and SOJA-RA$_{1\rightarrow 2}$ are the same from structural point of view (i.e., they have the same average depth, average width and number of nodes) since each cell of the base model H_1 is joined with exactly 1 cell of H_2.

In our experiments below, we use the summary tree quality σ_k (Section 3.3.4) to evaluate how good are our merged hierarchies.

Table 3.4 shows that σ_k values of the hierarchies provided by *UIA*$_{1in2}$, *UIA*$_{2in1}$ and *UAA*, when applied to (I_1, I_2), are in the same order of magnitude than that of the hierarchy provided by the centralized approach (*SEQ*).

Table 3.4 – σ_k-values - (I_1, I_2)

k	*SEQ*	*UIA1in2*	*UIA2in1*	*UAA*
2	0.61	0.56	0.54	0.55
4	0.59	0.57	0.56	0.54
6	0.56	0.55	0.57	0.53

Furthermore, observe that σ_k values (Table 3.5) of the hierarchies provided by SOJA$_{1\rightarrow 2}$ and SOJA-RA$_{1\rightarrow 2}$, when applied to (H_1, H_2), are comparable to that of hierarchies provided by *SEQ*. Moreover, notice that σ_k values of hierarchies provided by *TAJA* are greater than that of the hierarchies produced by *SEQ*. This is due to the fact that *SEQ*, as well as many existing incremental clustering algorithms, suffers from ordering effects. The first objects in an ordering

establish initial clusters that 'attract' the remaining objects and consequently, the quality of the generated clustering depends heavily on the initial choice of those clusters. This problem becomes more acute in the case of high dimensional data sets since it is common for all of the objects to be nearly equidistant from each other (curse of dimensionality [SEK03]), completely masking the well-defined initial clusters. However, *TAJA* joins well-defined clusters (from diverse areas of the object-description space) since the dimensionality of the data is reduced. Thus, each cluster of a joined partition is populated by objects that are close to each other and consequently, more similar on the full set of their attributes. On the other hand, objects belonging to different clusters of a joined partition are well separated regarding the full space since they are highly separated regarding the two subspaces.

Table 3.5 – σ_k-values - (H_1, H_2)

$\frac{N_1}{N}$	k	*SEQ*	SOJA$_{1\rightarrow 2}$	SOJA-RA$_{1\rightarrow 2}$	*TAJA*
0.20	3	0.61	0.59	0.57	0.68
0.20	7	0.59	0.57	0.56	0.64
0.50	3	0.61	0.60	0.59	0.65
0.50	7	0.59	0.55	0.53	0.63
0.80	3	0.61	0.58	0.54	0.67
0.80	7	0.59	0.58	0.52	0.66

Moreover, Table 3.6 shows that σ_k values of the hierarchies provided by SOJA$_{1\rightarrow 2}$ and SOJA-RA$_{1\rightarrow 2}$, when applied to (K_1, K_2), are also in the same order of magnitude than that of the hierarchy provided by the centralized approach. However, σ_k values of the hierarchies provided by *TAJA* decrease with the increase in the number of overlapping attributes $|Y|$.

Table 3.6 – σ_k-values - (K_1, K_2)

$\|Y\|$	k	*SEQ*	SOJA$_{1\rightarrow 2}$	SOJA-RA$_{1\rightarrow 2}$	*TAJA*
4	7	0.59	0.56	0.54	0.57
8	7	0.59	0.57	0.56	0.51
12	7	0.59	0.55	0.57	0.47

Thus, even if *UIA*, *UAA* , *SOJA*, *SOJA-RA* and *TAJA* provide clustering schemas with different structural properties (Figure 3.16, Figure 3.17, Figure 3.18 and Figure 3.19) than the one

provided by the centralized version of summary construction (*SEQ*), they are almost as effective as *SEQ*.

Finally, note that summary hierarchies are downsized versions of the original relation. Thus using them instead of the original data is expected to drastically reduce the data transmission cost between sites. More precisely, let H_R be a summary hierarchy of a database relation R having N attributes and containing n records. In the average-case assumption, the '*Compression Rate*' (CR) is given by :

$$CR = \frac{S_{H_R}}{S_R} = \frac{S_z \cdot (L + \frac{L-1}{d-1})}{S_t \cdot n}$$

where S_{H_R} (resp., S_R) is the average size in kilo-bytes of H_R (resp., R) and S_z (resp., S_t) the average size in kilo-bytes of summaries (resp., tuples) of H_R (resp., R). In the above formula, L is the number of cells of H_R and d its average width.

Recall that the number of leaves L is bounded by p^N (i.e., the size of the grid-based multi-dimensional space) where p represents the average number of descriptors defined for each attributes. However, the exact number greatly depends on the data set, and more specifically, on the existing correlations between attribute values. A reasonable assumption is $L = 0.1\% \cdot p^N$. Furthermore, each summary node contains the intentional description, the IDs (i.e., the extension) and some statistical information about incorporated tuples. Hence, S_z is only a small constant factor γ larger than S_t (i.e., $S_z = \gamma \cdot S_t$). Based on some real tests, $\gamma = 3$ is an acceptable approximation of the ratio $\frac{S_z}{S_t}$ [SPRM05]. For $p = 5$, $N = 8$ and $d = 3$, we find $CR \simeq \frac{1756}{n}$. Thus, in the case of large data set (e.g., $n >> 1756$), size reduction is clearly achieved by the SAINTETIQ process.

Those experimental results validate the theoretical assumptions that our merging algorithms are very efficient (Figure 3.13, Figure 3.14 and Figure 3.15) and result in high quality (Table 3.4, Table 3.5 and Table 3.6) clustering schemas.

3.5 Related work

The work presented in this chapter is closely related to both distributed clustering and tree alignment techniques. We briefly review both of them in this section.

Distributed Clustering

Clustering large data sets has recently emerged as an important area of research. The ever-increasing size of data sets and poor scalability of clustering algorithms has drawn attention to parallel and distributed clustering for partitioning large data sets. In [XJK99] a parallel version of DBSCAN and in [DM00] a parallel version of k-means were introduced. Both algorithms start with the complete data set residing on one central server and then distribute the data among different clients. But in the case of parallel k-means, the process requires synchronized communication during each iteration, which might become difficult and costly in a wide area network. Moreover, there might be constraints such as data could not be shared between different distributed locations due to privacy or security concerns about the data. There are some works where data in distributed environment has been clustered independently i.e. without any message passing among sources and multiple partitions combined using limited knowledge sharing ([SG02], [JN03], [GSM02], [GM03], [JKP03], and [KLM03]). Knowledge reuse framework [BG98] has also been explored, where label vectors of different partitions are combined without using any feature values ([SG02] and [JN03]). In [GSM02] and [GM03], distributed clustering has been discussed under two different settings that impose severe constraints on the nature of the data or knowledge shared by local data sites. In [JKP03], local sites are first clustered using the DBSCAN algorithm and then representatives from each local site are sent to a central site, where DBSCAN is applied again to find a global model. Another density estimation-based distributed clustering has been discussed in [KLM03]. In contrast to our proposal, these approaches do not offer a solution to the distributed hierarchical clustering problem ; they are based on partitioning techniques and generate flat clustering of the data.

In [SOGM02], Samatova et al. proposed RACHET, a clustering algorithm to handle horizontally distributed data. RACHET builds a global dendrogram by combining locally generated dendrograms using a hierarchical algorithm similar to our *UAA* algorithm. This algorithm uses statistical bounds to represent clusters efficiently across sites whereas we propose a grid-based approach with the use of linguistic variables and fuzzy partitions. In [JK99], Johnson et al. proposed a tree clustering approach to build a global dendrogram from individual dendrograms that are computed at local data sites over different sets of features. The algorithm first computes the element-wise intersection of all the most specialized clusters of sites to provide the most specialized partition of the whole data set, and then applies a single link clustering algorithm to compute the global dendrogram. This approach is similar to our *SEQ-MA* algorithm, but is less efficient since it is based on an agglomerative hierarchical clustering method. Indeed, its computational cost is $O(n^2)$, whereas for *SEQ-MA* it is $O(n \cdot \log n)$ where n is the size of the entire data

set. In [CSH03] (respectively [CSK04]), authors addressed decision tree (respectively bayesian network) learning from vertically distributed data. Both approaches are based on basic *directed* acyclic graph properties (the inner node specifies some test on a *single attribute*, the leaf node indicates the class, and the arc encodes conditional independencies between attributes). Since SAINTETIQ summaries are *multidimensional* and *unordered* trees, such algorithms cannot be used to merge them.

As far as we know, there are no multidimensional grid-based clustering algorithms for analyzing distributed data.

Tree Alignment

Graphs and trees are the most useful mathematical objects for representing any information with relational or hierarchical structure. Thus, they are common and well-studied combinatorial structures in computer science. One of the popular problems they pose is how to merge graphs together. It has been extensively studied and applied to several areas such as version control, schema/ontology integration, index merging, etc.

Keeping data synchronized across a variety of devices and environments raises the need for reconciliation of concurrently modified data. In [Lin04] and [Men02] documents are modelled as ordered trees and methods for merging them based on tree edit operations [Sel77] are outlined. Although there is a restricted approach [Zha96] that leads to merging unordered trees, one of the results from Zhang [ZJ94] is that finding the optimal matching function for such trees is an NP-complete problem. Since SAINTETIQ summaries are unordered trees and the constraints given in the definition of mappings in [Zha96] do not meet summary requirement (subtrees should be mapped to disjoint subtrees), such algorithms cannot be used to merge them.

Vast amount of data populate the internet and consequently the retrieval of relevant documents is often a non trivial task. Thus, schemas/ontologies matching ([KS03], [SE05]) is a critical operation in many application domains. It takes as input two schemas/ontologies, each consisting of a directed acyclic graph, and determines as output the relationships (equivalence, subsumption) holding between them. Most algorithms in this case, are based on various techniques (terminological, structural and semantic) and lead to identifying a mapping as a correspondence between close concepts using external resources like WordNet. In contrast to our proposal, these approaches focus on local mappings that need user evaluation and so cannot perform a total schemas/ontologies mapping.

Many scenarios in application and system maintenance require merging of two indexes : data migration in parallel database system, batch insertion in a centralized database system,

etc. A B-tree merging algorithm [SWSZ05] integrates two indexes covering the same key range into a single one. This algorithm is based on basic B-tree properties (balanced, high fanout, some minimal fill factor) and thus it cannot be used for the summaries merging issue where assumptions are less restrictive.

3.6 Conclusions

The SAINTETIQ summarization technique provides multi-resolution summaries (i.e., summary hierarchy) of structured data stored into a centralized database. However, in many real-world scenarios, data are often collected and stored in different databases located on distant sites and consequently the current centralized version of SAINTETIQ is either not feasible (resource limitations) or not desirable (privacy preserving).

In this Chapter, we proposed five algorithms for summarizing distributed data without a prior "unification" of the data sources : *Union by Incorporation Algorithm* (*UIA*) and *Union by Alignment Algorithm* (*UAA*) to deal with horizontally distributed data ; *Subspace-Oriented Join Algorithm* (*SOJA*), *SOJA with Rearrangements* (*SOJA-RA*) and *Tree Alignement-based Join Algorithm* (*TAJA*) to manage vertically distributed data. The main idea of these algorithms consists in applying innovative merges on two distinct summary hierarchies (local models) to provide a single summary hierarchy (global model). We showed that these algorithms provide a good-quality merged hierarchy while being very efficient in terms of computational time.

Conclusion and Perspectives

In this thesis, we have investigated a simple but useful strategy to alleviate the information overload often encountered by users when they access large and distributed Databases. In the following, we summarize our main contributions and outline some future directions of research.

Our main contributions are as follows :

- First, we proposed an efficient and effective algorithm coined Explore-Select-Rearrange Algorithm (*ESRA*) that uses database SAINTETIQ summaries to quickly provide users with concise, useful and structured representations of their query results. Given a user query, *ESRA* (i) explores the summary hierarchy (computed offline using SAINTETIQ) of the whole data stored in the database ; (ii) selects the most relevant summaries to that query ; (iii) rearranges them in a hierarchical structure based on the structure of the pre-computed summary hierarchy and (iv) returns the resulting hierarchy to the user. Each node (or summary) of the resulting hierarchy describes a subset of the result set in a user-friendly form using linguistic labels. The user then navigates through this hierarchy structure in a top-down manner, exploring the summaries of interest while ignoring the rest. The experimental results showed that the *ESRA* algorithm is efficient and provides well-formed (tight and clearly separated) and well-organized clusters of query results. Thus, it is very helpful to users who have vague and poorly defined retrieval goals or are interested in browsing through a set of items to explore what choices are available.
- The *ESRA* algorithm assumes that the summary hierarchy of the queried data is already built using SAINTETIQ and available as input. However, the SAINTETIQ model requires full access to the data which is going to be summarized, i.e., all data has to be located at the site where it is summarized. This requirement severely limits the applicability of the *ESRA* algorithm in a distributed environment, where data are distributed across many sites and their centralization cannot scale in terms of storage, computation and communication.

  The second contribution of this thesis is a solution for summarizing distributed data without a prior "unification" of the data sources. We assumed that the sources maintain their own summary hierarchies (local models), and we proposed new algorithms, namely *UIA*,

UAA, *SOJA*, *SOJA-RA* and *TAJA*, for merging them into a single final one (global model). *UIA* (*Union by Incorporation Algorithm*) and *UAA* (*Union by Alignment Algorithm*) deal with horizontally distributed data. They rely on a *union* operator that aggregates two input summaries (or clusters) representing two different sets of database records with the same set of attributes. *SOJA* (*Subspace-Oriented Join Algorithm*), *SOJA-RA* (*SOJA with Rearrangements*) and *TAJA* (*Tree Alignement-based Join Algorithm*) deal with vertically distributed data. They rely on a *join* operator that appends two input summaries (or clusters) representing the same set of database records with two different feature sets. We also proposed a consistent quality measure to quantify how good our merged hierarchies are. The experimental results showed that our merging algorithms result in high quality clustering schemas of the entire distributed data and are very efficient in terms of computational time.

Our future works are as follows :

- The *ESRA* algorithm does not exploit the statistical information (e.g., mean, maximum, minimum, standard deviation, tuple count) which is part of a summary's intent. The first future direction is to investigate the benefits that this statistical information could offer in the context of exploratory search and how these benefits can be achieved.
- The *ESRA* algorithm returns results using linguistic labels of summaries even if users express their queries using a free vocabulary. The second future direction is to study how to perform results rewriting when queries involve user-specific linguistic labels.
- The third future direction is to study criteria (other than time complexity) that allow us to choose the appropriate merging algorithm to use for a given data set.
- The fourth future direction is to study setting of the n-ary merge operator and provide (near-)optimal execution plans that order binary merges.
- Finally, we plan to extend our merging algorithms to deal with semantically heterogeneous summary hierarchies, i.e., summary hierarchies that are built using different *Knowledge Bases*.

Bibliography

[ACD02] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das.
Dbxplorer : enabling keyword search over relational databases.
In *SIGMOD '02 : Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 627–627, New York, NY, USA, 2002. ACM.

[All92] Dennis Allen.
Managing infoglut.
BYTE magazine, 17(6) :16, 1992.

[AMN+98] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu.
An optimal algorithm for approximate nearest neighbor searching fixed dimensions.
J. ACM, 45(6) :891–923, 1998.

[AMS+96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo.
Fast discovery of association rules.
In advances in knowledge discovery and data mining, pages 307–328, 1996.

[Ash94] M. Ashcraft.
Human Memory and Cognition.
Addison-Wesley Pub Co, 1994.

[AW00] Rakesh Agrawal and Edward L. Wimmers.
A framework for expressing and combining preferences.
SIGMOD Rec., 29(2) :297–306, 2000.

[Bay63] Rev. Thomas Bayes.
An essay toward solving a problem in the doctrine of chances.
Phil. Trans. Roy. Soc., 53 :370–418, 1763.

[BBK01] Christian Böhm, Stefan Berchtold, and Daniel A. Keim.
Searching in high-dimensional spaces : Index structures for improving the performance of multimedia databases.
ACM Comput. Surv., 33(3) :322–373, 2001.

[BBKK97] Stefan Berchtold, Christian Böhm, Daniel A. Keim, and Hans-Peter Kriegel.
A cost model for nearest neighbor search in high-dimensional data space.
In *PODS '97 : Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 78–86, New York, NY, USA, 1997. ACM.

[Ber01] Michael K. Bergman.
The deep web : Surfacing hidden value.
Journal of Electronic Publishing, 7(1), August 2001.

[Ber06] P. Berkhin.
A survey of clustering data mining techniques.
In Grouping Multidimensional Data : Recent Advances in Clustering, Ed. J. Kogan and C. Nicholas and M. Teboulle, pages 25–71, 2006.

[BG98] Kurt D. Bollacker and Joydeep Ghosh.

A supra-classifier architecture for scalable knowledge reuse.
In *Proc. 15th International Conf. on Machine Learning*, pages 64–72, 1998.

[BGL06] Indrajit Bhattacharya, Lise Getoor, and Louis Licamele.
Query-time entity resolution.
In *KDD '06*, pages 529–534, New York, NY, USA, 2006.

[BGRS99] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft.
When is "nearest neighbor" meaningful ?
In *ICDT '99 : Proceedings of the 7th International Conference on Database Theory*, pages 217–235, London, UK, 1999. Springer-Verlag.

[BH98] Krishna Bharat and Monika R. Henzinger.
Improved algorithms for topic distillation in a hyperlinked environment.
In *SIGIR '98 : Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 104–111, New York, NY, USA, 1998. ACM.

[BHN+02] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan.
Keyword searching and browsing in databases using banks.
In *ICDE*, pages 431–440, 2002.

[BK99] Yuri Breitbart and Henry F. Korth.
Replication and consistency in a distributed environment.
J. Comput. Syst. Sci., 59(1) :29–69, 1999.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel.
The x-tree : An index structure for high-dimensional data.
In *VLDB '96 : Proceedings of the 22th International Conference on Very Large Data Bases*, pages 28–39, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger.
The r*-tree : an efficient and robust access method for points and rectangles.
SIGMOD Rec., 19(2) :322–331, 1990.

[BMRB96] Bernadette Bouchon-Meunier, Maria Rifqi, and Sylvie Bothorel.
Towards general measures of comparison of objects.
Fuzzy Sets Syst., 84(2) :143–153, 1996.

[BP92] P. Bosc and O. Pivert.
Fuzzy querying in conventional databases.
In *Fuzzy logic for the management of uncertainty*, pages 645–671, New York, NY, USA, 1992. John Wiley & Sons, Inc.

[BP95] P. Bosc and O. Pivert.
Sqlf : a relational database language for fuzzy querying.
In *IEEE Transactions on Fuzzy Systems*, volume 3, pages 1–17, 1995.

[BP97] Patrick Bosc and Olivier Pivert.
Fuzzy queries against regular and fuzzy databases.
In *Flexible query answering systems*, pages 187–208, Norwell, MA, USA, 1997. Kluwer Academic Publishers.

[BP98] Sergey Brin and Lawrence Page.
The anatomy of a large-scale hypertextual web search engine.
Comput. Netw. ISDN Syst., 30(1-7) :107–117, 1998.

[BPR05] Patrick Bosc, Olivier Pivert, and Daniel Rocacher.
On the approximate division of fuzzy relations.
In *ISMIS*, pages 314–322, 2005.

[BPR07] P. Bosc, O. Pivert, and D. Rocacher.
About quotient and division of crisp and fuzzy relations.
J. Intell. Inf. Syst., 29(2) :185–210, 2007.

[BRM05] Bhuvan Bamba, Prasan Roy, and Mukesh Mohania.
Osqr : overlapping clustering of query results.
In *CIKM '05 : Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 239–240, New York, NY, USA, 2005. ACM.

[BRM07] Mounir Bechchi, Guillaume Raschia, and Noureddine Mouaddib.
Merging Distributed Database Summaries.
In *Conference on Information and Knowledge Management (CIKM)*, pages 419–428, Lisbon Portugal, 11 2007.

[BRM08] Mounir Bechchi, Guillaume Raschia, and Noureddine Mouaddib.
Joining Distributed Database Summaries.
Research Report RR-6768, INRIA, 2008.

[BRRT01] Allan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas.
Finding authorities and hubs from link structures on the world wide web.
In *WWW '01 : Proceedings of the 10th international conference on World Wide Web*, pages 415–429, New York, NY, USA, 2001. ACM.

[BVRM08] Mounir Bechchi, Amenel Voglozin, Guillaume Raschia, and Noureddine Mouaddib.
Multi-Dimensional Grid-Based Clustering of Fuzzy Query Results.
Research Report RR-6770, INRIA, 2008.

[BYRN99] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto.
Modern Information Retrieval.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[CCH92] James P. Callan, W. Bruce Croft, and Stephen M. Harding.
The inquery retrieval system.
In *Proceedings of the Third International Conference on Database and Expert Systems Applications*, pages 78–83. Springer-Verlag, 1992.

[CCH94] Wesley W. Chu, Qiming Chen, and Andy Hwang.
Query answering via cooperative data inference.
J. Intell. Inf. Syst., 3(1) :57–87, 1994.

[CCH04] Kaushik Chakrabarti, Surajit Chaudhuri, and Seung-won Hwang.
Automatic categorization of query results.
In *SIGMOD '04 : Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 755–766, New York, NY, USA, 2004. ACM.

[CCHY96] W. W. Chu, K. Chiang, C. Hsu, and H. Yau.
An error-based conceptual clustering method for providing approximate query answers.
Commun. ACM, 39 :216–230, 1996.

[CD03] Surajit Chaudhuri and Gautam Das.
Automated ranking of database query results.
In *CIDR*, pages 888–899, 2003.

[CDHW04] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum.
Probabilistic ranking of database query results.
In *Proceedings of the Thirtieth international conference on Very large data bases*, pages 888–899. VLDB Endowment, 2004.

[CHL+04] Kevin Chen-Chuan Chang, Bin He, Chengkai Li, Mitesh Patel, and Zhen Zhang.
Structured databases on the web : observations and implications.
SIGMOD Rec., 33(3) :61–70, 2004.

[Cho02] Jan Chomicki.
Querying with intrinsic preferences.
In *EDBT '02 : Proceedings of the 8th International Conference on Extending Database Technology*, pages 34–51, London, UK, 2002. Springer-Verlag.

[Cho03] Jan Chomicki.
Preference formulas in relational queries.
ACM Trans. Database Syst., 28(4) :427–466, 2003.

[CHZ05] Kevin Chen-Chuan Chang, Bin He, and Zhen Zhang.
Toward large scale integration : Building a metaquerier over databases on the web.
In *CIDR*, pages 44–55, 2005.

[Cia00] Pac nearest neighbor queries : Approximate and controlled search in high-dimensional and metric spaces.
In *ICDE '00 : Proceedings of the 16th International Conference on Data Engineering*, pages 244–255, Washington, DC, USA, 2000. IEEE Computer Society.

[CKVW06] David Cheng, Ravi Kannan, Santosh Vempala, and Grant Wang.
A divide-and-merge methodology for clustering.
ACM Trans. Database Syst., 31(4) :1499–1525, 2006.

[CQ69] A. Collins and M. Quillian.
Retrieval time from semantic memory.
Journal of Verbal Learning and Verbal Behavior, 8(2) :240–247, April 1969.

[Cro80] W. Bruce Croft.
A model of cluster searching based on classification.
Information Systems, 5 :189–195, 1980.

[CSH03] D. Caragea, A. Silvescu, and V. Honavar.
Decision tree induction from distributed heterogeneous autonomous data sources.
In *Proceedings of the International Conference on Intelligent Systems Design and Applications*, pages 341–350, 2003.

[CSK04] R. Chen, K. Sivakumar, and H. Kargupta.
Collective mining of bayesian networks from distributed heterogeneous data.
Knowl. Inf. Syst., 6(2) :164–187, 2004.

[CYC+96] Wesley W. Chu, Hua Yang, Kuorong Chiang, Michael Minock, Gladys Chow, and Chris Larson.
Cobase : a scalable and extensible cooperative information system.
J. Intell. Inf. Syst., 6(2-3) :223–259, 1996.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork.
Pattern Classification (2nd Edition).
Wiley-Interscience, November 2000.

[DKW02] Tran Khanh Dang, Josef Küng, and Roland Wagner.
Isa - an incremental hyper-sphere approach for efficiently solving complex vague queries.
In *DEXA '02 : Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 810–819, London, UK, 2002. Springer-Verlag.

[DM00] Inderjit S. Dhillon and Dharmendra S. Modha.
A data-clustering algorithm on distributed memory multiprocessors.
In *Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence*, pages 245–260, 2000.

[DPU02] Didier Dubois, Henri Prade, and Laurent Ughetto.
A new perspective on reasoning with fuzzy rules.
In *AFSS '02 : Proceedings of the 2002 AFSS International Conference on Fuzzy Systems. Calcutta*, pages 1–11, London, UK, 2002. Springer-Verlag.

[DXW+07] Bolin Ding, Xu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin.
Finding top-k min-cost connected trees in databases.
In *ICDE*, pages 836–845, 2007.

[FG05] Paolo Ferragina and Antonio Gulli.
A personalized search engine based on web-snippet hierarchical clustering.
In *WWW '05 : Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 801–810, New York, NY, USA, 2005. ACM.

[FGL98] Nir Friedman, Moisés Goldszmidt, and Thomas J. Lee.
Bayesian network classification with continuous attributes : Getting the best of both discretization and parametric fitting.
In *ICML '98 : Proceedings of the Fifteenth International Conference on Machine Learning*, pages 179–187, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[Fis87] Douglas H. Fisher.
Knowledge acquisition via incremental conceptual clustering.
Mach. Learn., 2(2) :139–172, 1987.

[FLN01] Ronald Fagin, Amnon Lotem, and Moni Naor.
Optimal aggregation algorithms for middleware.
In *PODS '01 : Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 102–113, New York, NY, USA, 2001. ACM.

[FSAA01] Hakan Ferhatosmanoglu, Ioana Stanoi, Divyakant Agrawal, and Amr El Abbadi.
Constrained nearest neighbor queries.
In *SSTD '01 : Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 257–278, London, UK, 2001. Springer-Verlag.

[FTAA01] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi.
Approximate nearest neighbor searching in multimedia databases.
In *ICDE*, pages 503–511, 2001.

[Gal08] Jose Galindo.
Handbook of Research on Fuzzy Information Processing in Databases.
Information Science Reference, Hershey, PA, 2008.

[GBK00] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling.
Optimizing multi-feature queries for image databases.
In *VLDB '00 : Proceedings of the 26th International Conference on Very Large Data Bases*, pages 419–428, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[GF04] David A. Grossman and Ophir Frieder.
Information Retrieval : Algorithms and Heuristics (The Kluwer International Series on Information Retrieval).
Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[GG98] Volker Gaede and Oliver Günther.
Multidimensional access methods.
ACM Comput. Surv., 30(2) :170–231, 1998.

[GGM94] Terry Gaasterland, Parke Godfrey, and Jack Minker.
An overview of cooperative answering.
In *Nonstandard queries and nonstandard answers : studies in logic and computation*, pages 1–40, Oxford, UK, 1994. Oxford University Press.

[GM03] Joydeep Ghosh and Srujana Merugu.
Distributed clustering with limited knowledge sharing.
In *Proceedings of the 5th International Conference on Advances in Pattern Recognition (ICAPR)*, pages 48–53, 2003.

[God97] Parke Godfrey.
Minimization in cooperative response to failing database queries.
IJCIS, 6(2) :95–149, 1997.

[GSM02] Joydeep Ghosh, Alexander Strehl, and Srujana Merugu.
A consensus framework for integrating distributed clusterings under limited knowledge sharing.
In *Proc. NSF Workshop on Next Generation Data Mining*, pages 99–108, November 2002.

[Gut88] Antonin Guttman.
R-trees : a dynamic index structure for spatial searching.
In *Readings in database systems*, pages 599–609, San Francisco, CA, USA, 1988. Morgan Kaufmann Publishers Inc.

[HBMB05] G. S. Halford, R. Baker, J. E. McCredden, and J. D. Bain.
How many variables can humans process ?
Psychological Science, 15 :70–76, 2005.

[HBV01] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis.
On clustering validation techniques.
Journal of Intelligent Information Systems, 17(2-3) :107–145, 2001.

[HGKI02] Taher H. Haveliwala, Aristides Gionis, Dan Klein, and Piotr Indyk.
Evaluating strategies for similarity search on the web.
In *WWW '02 : Proceedings of the 11th international conference on World Wide Web*, pages 432–442, New York, NY, USA, 2002. ACM.

[HGP03] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou.
Efficient ir-style keyword search over relational databases.
In *VLDB '2003 : Proceedings of the 29th international conference on Very large data bases*, pages 850–861. VLDB Endowment, 2003.

[HMYW04] Hai He, Weiyi Meng, Clement Yu, and Zonghuan Wu.
Automatic integration of web search interfaces with wise-integrator.
The VLDB Journal, 13(3) :256–273, 2004.

[HP96] Marti A. Hearst and Jan O. Pedersen.

Reexamining the cluster hypothesis : scatter/gather on retrieval results.
In *SIGIR '96 : Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 76–84, New York, NY, USA, 1996. ACM.

[HP02] Vagelis Hristidis and Yannis Papakonstantinou.
Discover : keyword search in relational databases.
In *VLDB '02 : Proceedings of the 28th international conference on Very Large Data Bases*, pages 670–681. VLDB Endowment, 2002.

[HTF01] T. Hastie, R. Tibshirani, and J. H. Friedman.
The Elements of Statistical Learning.
Springer, 2nd edition, August 2001.

[HW79] J. A. Hartigan and M. A. Wong.
A K-means clustering algorithm.
Applied Statistics, 28 :100–108, 1979.

[HWYY07] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu.
Blinks : ranked keyword searches on graphs.
In *SIGMOD '07 : Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 305–316, New York, NY, USA, 2007. ACM.

[IBS08] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman.
A survey of top-k query processing techniques in relational database systems.
ACM Comput. Surv., 40(4) :1–58, 2008.

[IH86] Tadao Ichikawa and Masahito Hirakawa.
Ares : a relational database with the capability of performing flexible interpretation of queries.
IEEE Trans. Softw. Eng., 12(5) :624–634, 1986.

[IM98] Piotr Indyk and Rajeev Motwani.
Approximate nearest neighbors : towards removing the curse of dimensionality.
In *STOC '98 : Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, New York, NY, USA, 1998. ACM.

[JK99] Erik L. Johnson and Hillol Kargupta.
Collective, hierarchical clustering from distributed, heterogeneous data.
In *Large-Scale Parallel Data Mining*, pages 221–244, 1999.

[JKP03] Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle.
Towards effective and efficient distributed clustering.
In *Proc. Int. Workshop on Clustering Large Data Sets, 3rd IEEE International Conference on Data Mining (ICDM)*, pages 49–58, 2003.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn.
Data clustering : a review.
ACM Comput. Surv., 31(3) :264–323, September 1999.

[JN03] Pierre E. Jouve and Nicolas Nicoloyannis.
A new method for combining partitions, applications for distributed clustering.
In *International Workshop on Paralell and Distributed Machine Learning and Data Mining (ECML/PKDD03)*, pages 35–46, September 2003.

[JR71] N. Jardine and C. J. Van Rijsbergen.

The use of hierarchical clustering in information retrieval.
Information Storage and Retrieval, 7 :217–240, 1971.

[Kap83] S. J. Kaplan.
Cooperative responses from a portable natural language database query system.
In M. Brady and R. C. Berwick, editors, *Computational Models of Discourse*, pages 167–208. MIT Press, Cambridge, MA, 1983.

[KBL07] Dimitre Kostadinov, Mokrane Bouzeghoub, and Stéphane Lopes.
Query rewriting based on user's profile knowledge.
In *BDA*, 2007.

[KDH+07] Nishant Kapoor, Gautam Das, Vagelis Hristidis, S. Sudarshan, and Gerhard Weikum.
Star : A system for tuple and attribute ranking of query answers.
In *ICDE*, pages 1483–1484, 2007.

[KI05] Georgia Koutrika and Yannis Ioannidis.
A unified user profile framework for query disambiguation and personalization.
In *Proceedings of Workshop on New Technologies for Personalized Information Access*, pages 44–53, July 2005.

[Kie02] Werner Kießling.
Foundations of preferences in database systems.
In *VLDB '02 : Proceedings of the 28th international conference on Very Large Data Bases*, pages 311–322. VLDB Endowment, 2002.

[Kle99] Jon M. Kleinberg.
Authoritative sources in a hyperlinked environment.
J. ACM, 46(5) :604–632, 1999.

[KLM03] M. Klusch, S. Lodi, and G. Moro.
Distributed clustering based on sampling local density estimates.
In *IJCAI*, pages 485–490, 2003.

[KM92] Jyrki Kivinen and Heikki Mannila.
Approximate dependency inference from relations.
In *ICDT '92 : Proceedings of the 4th International Conference on Database Theory*, pages 86–98, London, UK, 1992. Springer-Verlag.

[KMP00] E. P. Klement, R. Mesiar, and E. Pap.
Triangular norms.
Kluwer Academic Publishers, 2000.

[KPC+05] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar.
Bidirectional expansion for keyword search on graph databases.
In *VLDB '05 : Proceedings of the 31st international conference on Very large data bases*, pages 505–516. VLDB Endowment, 2005.

[KS03] Yannis Kalfoglou and Marco Schorlemmer.
Ontology mapping : the state of the art.
Knowl. Eng. Rev., 18(1) :1–31, 2003.

[KS06] Benny Kimelfeld and Yehoshua Sagiv.
Finding and approximating top-k answers in keyword proximity search.
In *PODS '06 : Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 173–182, New York, NY, USA, 2006. ACM.

[Lew96] David Lewis.
Dying for information : An investigation into the effects of information overload in the usa and worldwide.
Reuters Limited, 1996.

[Lin04] Tancred Lindholm.
A three-way merge for xml documents.
In *DocEng '04 : Proceedings of the 2004 ACM symposium on Document engineering*, pages 1–10, New York, NY, USA, 2004. ACM.

[LL87] M. Lacroix and Pierre Lavency.
Preferences ; putting more knowledge into queries.
In *VLDB '87 : Proceedings of the 13th International Conference on Very Large Data Bases*, pages 217–225, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[LWL+07] Chengkai Li, Min Wang, Lipyeow Lim, Haixun Wang, and Kevin Chen-Chuan Chang.
Supporting ranking and clustering as generalized order-by and group-by.
In *SIGMOD '07 : Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 127–138, New York, NY, USA, 2007. ACM.

[LYMC06] Fang Liu, Clement Yu, Weiyi Meng, and Abdur Chowdhury.
Effective keyword search in relational databases.
In *SIGMOD '06 : Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 563–574, New York, NY, USA, 2006. ACM.

[Man67] G. Mandler.
Organization in memory.
In K. W. Spense and J. T. Spense, editors, *The psychology of learning and motivation*, pages 327–372. Academic Press, 1967.

[MBKB02] Sean D. MacArthur, Carla E. Brodley, Avinash C. Ka, and Lynn S. Broderick.
Interactive content-based image retrieval using relevance feedback.
Comput. Vis. Image Underst., 88(2) :55–75, 2002.

[MC93] Matthew Merzbacher and Wesley W. Chu.
Pattern-based clustering for database attribute values.
In *Proceedings of AAAI Workshop on Knowledge Discovery*, pages 1–8, 1993.

[Men02] T. Mens.
A state-of-the-art survey on software merging.
IEEE Trans. Softw. Eng., 28(5) :449–462, 2002.

[Mil56] George A. Miller.
The magical number seven, plus or minus two : Some limits on our capacity for processing information.
The Psychological Review, 63 :81–97, 1956.

[Mil62] G. A. Miller.
Information input overload.
In M. C. Yovits, G. T. Jacobi, and G. D. Goldstein, editors, *Conference on Self-Organizing Systems*. Spartan Books, 1962.

[ML05] I. Muslea and T. Lee.
Online query relaxation via bayesian causal structures discovery.
In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, pages 831–836, 2005.

[Mot86a] Amihai Motro.
Query generalization : a method for interpreting null answers.
In *Proceedings from the first international workshop on Expert database systems*, pages 597–616, Redwood City, CA, USA, 1986. Benjamin-Cummings Publishing Co., Inc.

[Mot86b] Amihai Motro.
Seave : a mechanism for verifying user presuppositions in query systems.
ACM Trans. Inf. Syst., 4(4) :312–330, 1986.

[Mot88] Amihai Motro.
Vague : a user interface to relational databases that permits vague queries.
ACM Trans. Inf. Syst., 6(3) :187–214, 1988.

[MR05] Oded Maimon and Lior Rokach, editors.
The Data Mining and Knowledge Discovery Handbook.
Springer, 2005.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze.
Introduction to Information Retrieval.
Cambridge University Press, New York, NY, USA, 2008.

[Mus04] Ion Muslea.
Machine learning for online query relaxation.
In *KDD '04 : Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 246–255, New York, NY, USA, 2004. ACM.

[Nam05] Ullas Nambiar.
Answering imprecise queries over autonomous databases.
PhD thesis, Tempe, AZ, USA, 2005.
Adviser-Kambhampati, Subrarao.

[Nie03] Jakob Nielsen.
Curmudgeon : Im, not ip (information pollution).
ACM Queue, 1(8) :76–75, 2003.

[NK03] Ullas Nambiar and Subbarao Kambhampati.
Answering imprecise database queries : a novel approach.
In *WIDM '03 : Proceedings of the 5th ACM international workshop on Web information and data management*, pages 126–133, New York, NY, USA, 2003. ACM.

[NK04] Ullas Nambiar and Subbarao Kambhampati.
Mining approximate functional dependencies and concept similarities to answer imprecise queries.
In *WebDB '04 : Proceedings of the 7th International Workshop on the Web and Databases*, pages 73–78, New York, NY, USA, 2004. ACM.

[NR99] Surya Nepal and M. V. Ramakrishna.
Query processing issues in image (multimedia) databases.
In *Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Austrialia*, pages 22–29. IEEE Computer Society, 1999.

[NZJ01] Andrew Y. Ng, Alice X. Zheng, and Michael I. Jordan.
Stable algorithms for link analysis.
In *SIGIR '01 : Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 258–266, New York, NY, USA, 2001. ACM.

[OV99] M. Tamer Özsu and Patrick Valduriez.
Principles of distributed database systems (2nd ed.).
Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[Qui68] M. R. Quillian.
Semantic memory.
In Minsky M., editor, *Semantic Information Processing*, pages 227–270. The MIT Press, 1968.

[Qui93] J. Ross Quinlan.
C4.5 : programs for machine learning.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[RKV95] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent.
Nearest neighbor queries.
In *SIGMOD '95 : Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 71–79, New York, NY, USA, 1995. ACM.

[RM75] E. Rosch and C.B. Mervis.
Family ressemblances : studies in the internal structure of categories.
Cognitive Psychology, 7 :573–605, 1975.

[RM02] G. Raschia and N. Mouaddib.
SAINTETIQ : a fuzzy set-based approach to database summarization.
Fuzzy Sets Syst., 129(2) :137–162, 2002.

[Rob97] S. E. Robertson.
The probability ranking principle in ir.
In *Readings in information retrieval*, pages 281–286, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[Rus69] E. Ruspini.
A new approach to clustering.
Information and Control, 15 :22–32, 1969.

[Sal89] Gerard Salton.
Automatic text processing : the transformation, analysis, and retrieval of information by computer.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[SCZ00] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang.
Wavecluster : a wavelet-based clustering approach for spatial data in very large databases.
The VLDB Journal, 8(3-4) :289–304, 2000.

[SD06] Parag Singla and Pedro Domingos.
Entity resolution with markov logic.
In *ICDM '06*, pages 572–582, Washington, DC, USA, 2006.

[SE98] Erich Schikuta and Martin Erhart.
Bang-clustering : A novel grid-clustering algorithm for huge data sets.
In *SSPR '98/SPR '98 : Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, pages 867–874, London, UK, 1998. Springer-Verlag.

[SE05] Pavel Shvaiko and Jérôme Euzenat.
A survey of schema-based matching approaches.
In *J. Data Semantics IV*, pages 146–171, 2005.

[SEK03] Michael Steinbach, Levent Ertöz, and Vipin Kumar.
The challenges of clustering high-dimensional data.
In *New Vistas in Statistical Physics : Applications in Econophysics, Bioinformatics, and Pattern Recognition*, pages 273–307. Springer-Verlag, 2003.

[Sel77] S. M. Selkow.
The tree-to-tree editing problem.
Information processing letters, pages 184–186, 1977.

[SG02] Alexander Strehl and Joydeep Ghosh.
Cluster ensembles – a knowledge reuse framework for combining multiple partitions.
Journal on Machine Learning Research (JMLR), 3 :583–617, December 2002.

[She97] David Shenk.
Data Smog : Surviving the Information Glut.
HarperCollins Publishers, New York, NY, USA, 1997.

[Sim74] Herbert A. Simon.
How big is a chunk ?
Science, 183 :482–488, 1974.

[SK05] Marc Shapiro and Nishith Krishna.
The three dimensions of data consistency.
In *Journées Francophones sur la Cohérence des Données en Univers Réparti (CDUR*, pages 54–58, 2005.

[SM86] Gerard Salton and Michael J. McGill.
Introduction to Modern Information Retrieval.
McGraw-Hill, Inc., New York, NY, USA, 1986.

[SOGM02] Nagiza F. Samatova, George Ostrouchov, Al Geist, and Anatoli V. Melechko.
Rachet : An efficient cover-based merging of clustering hierarchies from distributed datasets.
Distrib. Parallel Databases, 11(2) :157–180, 2002.

[SPRM05] Regis Saint-Paul, Guillaume Raschia, and Noureddine Mouaddib.
General purpose database summarization.
In *VLDB '05*, pages 733–744, 2005.

[SV98] Ulrike Schultz and Betty Vandenbosch.
Information overload in a groupware environment : Now you see it, now you don't.
Journal of Organizational Computing and Electronic Commerce, 8(2) :127–148, 1998.

[SWHL06] Weifeng Su, Jiying Wang, Qiong Huang, and Fred Lochovsky.
Query result ranking over e-commerce web databases.
In *CIKM '06 : Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 575–584, New York, NY, USA, 2006. ACM.

[SWSZ05] Xiaowei Sun, Rui Wang, Betty Salzberg, and Chendong Zou.
Online b-tree merging.
In *SIGMOD '05 : Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 335–346, New York, NY, USA, 2005.

[Tah77] V. Tahani.
A conceptual framework for fuzzy query processing : A step toward very intelligent database systems.
Information Processing and Management, 13 :289–303, 1977.

[Tve77] A. Tversky.
Features of similarity.
Psycol. Rev., 84(4) :327–352, 1977.

[Voo85] Ellen M. Voorhees.
The cluster hypothesis revisited.
In *SIGIR '85 : Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 188–196, New York, NY, USA, 1985. ACM.

[vRC75] C. J. van Rijsbergen and W. Bruce Croft.
Document clustering : An evaluation of some experiments with the cranfield 1400 collection.
Inf. Process. Manage., 11(5-7) :171–182, 1975.

[VRUM04] W.A. Voglozin, G. Raschia, L. Ughetto, and N. Mouaddib.
Querying the SaintEtiQ summaries - a first attempt.
In *6th International Conference on Flexible Query Answering Systems*, pages 404–417, Lyon, France, June 24-26 2004.

[WB00] Roger Weber and Klemens Böhm.
Trading quality for time with nearest neighbor search.
In *EDBT '00 : Proceedings of the 7th International Conference on Extending Database Technology*, pages 21–35, London, UK, 2000. Springer-Verlag.

[WFSP00] Leejay Wu, Christos Faloutsos, Katia P. Sycara, and Terry R. Payne.
Falcon : Feedback adaptive loop for content-based retrieval.
In *VLDB '00 : Proceedings of the 26th International Conference on Very Large Data Bases*, pages 297–306, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[Win98] William Van Winkle.
Information overload : Fighting data asphyxiation is difficult but possible.
Computer Bits magazine, 8(2), 1998.

[WJ96] David A. White and Ramesh Jain.
Similarity indexing with the ss-tree.
In *ICDE '96 : Proceedings of the Twelfth International Conference on Data Engineering*, pages 516–523, Washington, DC, USA, 1996. IEEE Computer Society.

[WR97] Michelle M. Weil and Larry D. Rosen.
TechnoStress : Coping With Technology Work Home Play.
John Wiley and Sons, New York, NY, USA, 1997.

[WYM97] Wei Wang, Jiong Yang, and Richard R. Muntz.
STING : A statistical information grid approach to spatial data mining.
In *Twenty-Third International Conference on Very Large Data Bases*, pages 186–195, Athens, Greece, 1997. Morgan Kaufmann.

[XJK99] Xiaowei Xu, Jochen Jager, and Hans-Peter Kriegel.
A fast parallel clustering algorithm for large spatial databases.
Data Min. Knowl. Discov., 3(3) :263–290, 1999.

[YM98] Clement T. Yu and Weiyi Meng.
Principles of database query processing for advanced applications.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[Zad56] L. A. Zadeh.
Fuzzy sets.
Information and Control, 8 :338–353, 1956.

[Zad75] L. A. Zadeh.
Concept of a linguistic variable and its application to approximate reasoning.
Information and Syst., 1 :119–249, 1975.

[Zad99] L. A. Zadeh.
Fuzzy sets as a basis for a theory of possibility.
Fuzzy Sets Syst., 100 :9–34, 1999.

[ZE99] Oren Zamir and Oren Etzioni.
Grouper : a dynamic clustering interface to web search results.
In *WWW '99 : Proceedings of the eighth international conference on World Wide Web*, pages 1361–1374, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[Zha96] Kaizhong Zhang.
A constrained edit distance between unordered labeled trees.
Algorithmica, 15(3) :205–222, 1996.

[ZHC+04] Hua-Jun Zeng, Qi-Cai He, Zheng Chen, Wei-Ying Ma, and Jinwen Ma.
Learning to cluster web search results.
In *SIGIR '04 : Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 210–217, New York, NY, USA, 2004. ACM.

[ZJ94] Kaizhong Zhang and Tao Jiang.
Some max snp-hard results concerning unordered labeled trees.
Inf. Process. Lett., 49(5) :249–254, 1994.

[ZK96] Slawomir Zadrozny and Janusz Kacprzyk.
Fquery for access : towards human consistent querying user interface.
In *SAC '96 : Proceedings of the 1996 ACM symposium on Applied Computing*, pages 532–536, New York, NY, USA, 1996. ACM.

[Zlo75] Moshé M. Zloof.
Query-by-example : the invocation and definition of tables and forms.
In *VLDB '75 : Proceedings of the 1st International Conference on Very Large Data Bases*, pages 1–24, New York, NY, USA, 1975. ACM.

Clustering-based Approximate Answering of Query Result in Large and Distributed Databases

Mounir BECHCHI

Abstract

Database systems are increasingly used for interactive and exploratory data retrieval. In such retrievals, users queries often result in too many answers, so users waste significant time and efforts sifting and sorting through these answers to find the relevant ones. In this thesis, we first propose an efficient and effective algorithm coined Explore-Select-Rearrange Algorithm (*ESRA*), based on the SAINTETIQ model, to quickly provide users with hierarchical clustering schemas of their query results. SAINTETIQ is a domain knowledge-based approach that provides multi-resolution summaries of structured data stored into a database. Each node (or summary) of the hierarchy provided by *ESRA* describes a subset of the result set in a user-friendly form based on domain knowledge. The user then navigates through this hierarchy structure in a top-down fashion, exploring the summaries of interest while ignoring the rest. Experimental results show that the *ESRA* algorithm is efficient and provides well-formed (tight and clearly separated) and well-organized clusters of query results. The *ESRA* algorithm assumes that the summary hierarchy of the queried data is already built using SAINTETIQ and available as input. However, SAINTETIQ requires full access to the data which is going to be summarized. This requirement severely limits the applicability of the *ESRA* algorithm in a distributed environment, where data is distributed across many sites and transmitting the data to a central site is not feasible or even desirable. The second contribution of this thesis is therefore a solution for summarizing distributed data without a prior "unification" of the data sources. We assume that the sources maintain their own summary hierarchies (local models), and we propose new algorithms for merging them into a single final one (global model). An experimental study shows that our merging algorithms result in high quality clustering schemas of the entire distributed data and are very efficient in terms of computational time.

Keywords : Relational databases, Database summaries (the SAINTETIQ model), Clustering of query results, Distributed clustering.

Réponses Approchées de Résultats de Requêtes par Classification dans des Bases de Données Volumineuses et Distribuées

Résumé

Les utilisateurs des bases de données doivent faire face au problème de surcharge d'information lors de l'interrogation de leurs données, qui se traduit par un nombre de réponses trop élevé à des requêtes exploratoires. Pour remédier à ce problème, nous proposons un algorithme efficace et rapide, appelé *ESRA* (Explore-Select-Rearrange Algorithm), qui utilise les résumés SAINTETIQ pré-calculés sur l'ensemble des données pour regrouper les réponses à une requête utilisateur en un ensemble de classes (ou résumés) organisées hiérarchiquement. Chaque classe décrit un sous-ensemble de résultats dont les propriétés sont voisines. L'utilisateur pourra ainsi explorer la hiérarchie pour localiser les données qui l'intéressent et en écarter les autres. Les résultats expérimentaux montrent que l'algorithme *ESRA* est efficace et fournit des classes bien formées (i.e., leur nombre reste faible et elles sont bien séparées). Cependant, le modèle SAINTETIQ, utilisé par l'algorithme *ESRA*, exige que les données soient disponibles sur le serveur des résumés. Cette hypothèse rend inapplicable l'algorithme *ESRA* dans des environnements distribués où il est souvent impossible ou peu souhaitable de rassembler toutes les données sur un même site. Pour remédier à ce problème, nous proposons une collection d'algorithmes qui combinent deux résumés générés localement et de manière autonome sur deux sites distincts pour en produire un seul résumant l'ensemble des données distribuées, sans accéder aux données d'origine. Les résultats expérimentaux montrent que ces algorithmes sont aussi performants que l'approche centralisée (i.e., SAINTETIQ appliqué aux données après regroupement sur un même site) et produisent des hiérarchies très semblables en structure et en qualité à celles produites par l'approche centralisée.

Mots-clés: Base de données relationnelles, Résumés de données (Le modèle SAINTETIQ), Classification des résultats de requêtes, Classification distribuée.

Discipline : Informatique
Spécialité : Bases de Données

Laboratoire : LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE.
UMR CNRS 6241. 2, rue de la Houssinière, BP 92 208 – 44 322 Nantes, CEDEX 3.